

## TCD1304 Controller

Generated by Doxygen 1.13.2



<b>1 Firmware for the TCD1304 using FlexPWM</b>	<b>1</b>
1.0.1 In this directory:	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 TCD1304Device::Frame_Header_struct Struct Reference	7
4.1.1 Member Data Documentation	7
4.1.1.1 avgdummy	7
4.1.1.2 buffer	8
4.1.1.3 error_flag	8
4.1.1.4 frame_counter	8
4.1.1.5 frame_elapsed_secs	8
4.1.1.6 frame_exposure_secs	8
4.1.1.7 frames_completed	8
4.1.1.8 frameset_counter	8
4.1.1.9 framesets_completed	8
4.1.1.10 mode	8
4.1.1.11 nbuffer	8
4.1.1.12 offset	9
4.1.1.13 oops_flag	9
4.1.1.14 ready_for_send	9
4.1.1.15 timer_difference_secs	9
4.1.1.16 timer_elapsed_secs	9
4.1.1.17 trigger_counter	9
4.1.1.18 trigger_difference_secs	9
4.1.1.19 trigger_elapsed_secs	9
4.1.1.20 trigger_mode	9
4.2 TCD1304Device::SubModule Struct Reference	10
4.2.1 Constructor & Destructor Documentation	10
4.2.1.1 SubModule()	10
4.2.2 Member Data Documentation	11
4.2.2.1 ctrl2_mask	11
4.2.2.2 divider	11
4.2.2.3 filler	11
4.2.2.4 flexpwm	11

4.2.2.5 inten_mask	11
4.2.2.6 intena_mask	11
4.2.2.7 invertA	11
4.2.2.8 invertB	11
4.2.2.9 irq	11
4.2.2.10 isr	12
4.2.2.11 mask	12
4.2.2.12 muxvalA	12
4.2.2.13 muxvalB	12
4.2.2.14 name	12
4.2.2.15 newvals	12
4.2.2.16 offA_counts	12
4.2.2.17 offB_counts	12
4.2.2.18 onA_counts	12
4.2.2.19 onB_counts	12
4.2.2.20 period_counts	13
4.2.2.21 period_secs	13
4.2.2.22 pinA	13
4.2.2.23 pinB	13
4.2.2.24 prescale	13
4.2.2.25 submod	13
4.3 TCD1304Device Class Reference	14
4.3.1 Member Typedef Documentation	18
4.3.1.1 Frame_Header	18
4.3.2 Constructor & Destructor Documentation	18
4.3.2.1 TCD1304Device()	18
4.3.3 Member Function Documentation	19
4.3.3.1 attach_isr()	19
4.3.3.2 check_submodule()	19
4.3.3.3 clear_busypin()	20
4.3.3.4 clear_error_flags()	20
4.3.3.5 clear_frames_completed_callback()	20
4.3.3.6 clear_framesets_completed_callback()	20
4.3.3.7 clear_ldok()	20
4.3.3.8 clear_mode()	20
4.3.3.9 clear_run()	21
4.3.3.10 clear_sync_busy_pins()	21
4.3.3.11 clear_syncpin()	21
4.3.3.12 close()	21

---

4.3.3.13 cycles64()	22
4.3.3.14 disable_irqs()	22
4.3.3.15 fill_frame_header()	22
4.3.3.16 flexpwm_start()	23
4.3.3.17 flexpwm_stop()	24
4.3.3.18 flexpwm_wait()	24
4.3.3.19 force()	24
4.3.3.20 frameset_arm()	24
4.3.3.21 frameset_cnvst_isr()	25
4.3.3.22 frameset_icg_isr()	25
4.3.3.23 frameset_init_frames()	25
4.3.3.24 frameset_init_frameset()	26
4.3.3.25 frameset_sh_isr()	27
4.3.3.26 frameset_start()	27
4.3.3.27 load_frames_completed_callback()	28
4.3.3.28 load_framesets_completed_callback()	28
4.3.3.29 load_submodule()	28
4.3.3.30 print_and_check_submodule()	29
4.3.3.31 print_counters()	29
4.3.3.32 print_errormsg() [1/2]	30
4.3.3.33 print_errormsg() [2/2]	30
4.3.3.34 print_submodule()	30
4.3.3.35 printbits16_()	31
4.3.3.36 pulse_arm()	31
4.3.3.37 pulse_cnvst_isr()	31
4.3.3.38 pulse_icg_isr()	32
4.3.3.39 pulse_init_frames()	32
4.3.3.40 pulse_init_frameset()	32
4.3.3.41 pulse_sh_isr()	33
4.3.3.42 pulse_start()	34
4.3.3.43 read() [1/2]	34
4.3.3.44 read() [2/2]	35
4.3.3.45 register_dump()	36
4.3.3.46 set_clock_master() [1/2]	37
4.3.3.47 set_clock_master() [2/2]	37
4.3.3.48 set_clock_slave() [1/2]	37
4.3.3.49 set_clock_slave() [2/2]	38
4.3.3.50 set_clock_sync() [1/2]	38
4.3.3.51 set_clock_sync() [2/2]	39

---

4.3.3.52 set_init()	39
4.3.3.53 set_ldok()	39
4.3.3.54 set_outen()	39
4.3.3.55 set_outen_off()	39
4.3.3.56 set_outen_on()	40
4.3.3.57 set_outenA_on()	40
4.3.3.58 set_prescale()	40
4.3.3.59 set_run()	40
4.3.3.60 set_val0()	40
4.3.3.61 set_val1()	40
4.3.3.62 set_val2()	40
4.3.3.63 set_val3()	41
4.3.3.64 set_val4()	41
4.3.3.65 set_val5()	41
4.3.3.66 setup_digital_pins()	41
4.3.3.67 setup_frameset()	42
4.3.3.68 setup_pulse()	43
4.3.3.69 setup_submodule()	43
4.3.3.70 setup_timer()	44
4.3.3.71 setup_triggers()	45
4.3.3.72 sh_difference_secs()	46
4.3.3.73 sh_elapsed_secs()	46
4.3.3.74 sh_exposure_secs()	46
4.3.3.75 start_read()	47
4.3.3.76 start_triggers()	47
4.3.3.77 stop_all()	48
4.3.3.78 stop_runs_only()	48
4.3.3.79 stop_triggers()	49
4.3.3.80 stop_with_irqs()	49
4.3.3.81 timer_difference_secs()	50
4.3.3.82 timer_elapsed_secs()	50
4.3.3.83 timer_isr()	50
4.3.3.84 timer_start()	51
4.3.3.85 timer_stop()	52
4.3.3.86 timer_stop_with_irq()	52
4.3.3.87 timer_wait()	52
4.3.3.88 toggle_busypin()	53
4.3.3.89 toggle_syncpin()	53
4.3.3.90 trigger_difference_secs()	53

4.3.3.91 trigger_elapsed_secs()	53
4.3.3.92 trigger_isr()	54
4.3.3.93 triggered_read() [1/2]	54
4.3.3.94 triggered_read() [2/2]	55
4.3.3.95 update_read_buffer()	56
4.3.3.96 wait_read()	56
4.3.3.97 wait_triggered_read()	57
4.3.3.98 wait_triggers()	57
4.3.4 Member Data Documentation	58
4.3.4.1 busytoggled	58
4.3.4.2 clk	58
4.3.4.3 cnvst	58
4.3.4.4 cnvst_counter	58
4.3.4.5 cnvst_extra_delay_counts	58
4.3.4.6 error_flag	58
4.3.4.7 flexpwm	58
4.3.4.8 flexpwm_running	58
4.3.4.9 frame_counter	58
4.3.4.10 frame_counts	59
4.3.4.11 frames_completed_callback	59
4.3.4.12 frameset_armed	59
4.3.4.13 frameset_counter	59
4.3.4.14 frameset_counts	59
4.3.4.15 framesets_completed_callback	59
4.3.4.16 icg	59
4.3.4.17 icg_counter	59
4.3.4.18 mode	59
4.3.4.19 oops_flag	60
4.3.4.20 pulse_armed	60
4.3.4.21 read_buffer	60
4.3.4.22 read_callback	60
4.3.4.23 read_counter	60
4.3.4.24 read_counts	60
4.3.4.25 read_expected_time	60
4.3.4.26 read_pointer	60
4.3.4.27 sh	60
4.3.4.28 sh_clearing_counter	61
4.3.4.29 sh_clearing_counts	61
4.3.4.30 sh_counter	61

4.3.4.31 sh_counts_per_icg . . . . .	61
4.3.4.32 sh_cyccnt64_exposure . . . . .	61
4.3.4.33 sh_cyccnt64_now . . . . .	61
4.3.4.34 sh_cyccnt64_prev . . . . .	61
4.3.4.35 sh_cyccnt64_start . . . . .	61
4.3.4.36 sh_short_period_counts . . . . .	61
4.3.4.37 skip_one . . . . .	61
4.3.4.38 skip_one_reload . . . . .	62
4.3.4.39 sync_enabled . . . . .	62
4.3.4.40 sync_pin . . . . .	62
4.3.4.41 synctoggled . . . . .	62
4.3.4.42 timer . . . . .	62
4.3.4.43 timer_callback . . . . .	62
4.3.4.44 timer_cyccnt64_now . . . . .	62
4.3.4.45 timer_cyccnt64_prev . . . . .	62
4.3.4.46 timer_cyccnt64_start . . . . .	62
4.3.4.47 timer_first_time_flag . . . . .	63
4.3.4.48 timer_inner_counter . . . . .	63
4.3.4.49 timer_inner_counts . . . . .	63
4.3.4.50 timer_interframe_min_secs . . . . .	63
4.3.4.51 timer_interval_secs . . . . .	63
4.3.4.52 timer_outer_counter . . . . .	63
4.3.4.53 timer_outer_counts . . . . .	63
4.3.4.54 timer_period_secs . . . . .	63
4.3.4.55 timer_running . . . . .	63
4.3.4.56 timerflexpwm . . . . .	63
4.3.4.57 trigger_attached . . . . .	64
4.3.4.58 trigger_busy . . . . .	64
4.3.4.59 trigger_callback . . . . .	64
4.3.4.60 trigger_counter . . . . .	64
4.3.4.61 trigger_counts . . . . .	64
4.3.4.62 trigger_cyccnt64_now . . . . .	64
4.3.4.63 trigger_cyccnt64_prev . . . . .	64
4.3.4.64 trigger_cyccnt64_start . . . . .	64
4.3.4.65 trigger_edge_mode . . . . .	64
4.3.4.66 trigger_mode . . . . .	64
4.3.4.67 trigger_pin . . . . .	65
4.3.4.68 trigger_pin_mode . . . . .	65
4.4 usb_string_descriptor_struct_manufacturer Struct Reference . . . . .	65



4.4.1 Member Data Documentation	65
4.4.1.1 bDescriptorType	65
4.4.1.2 bLength	65
4.4.1.3 wString	65
4.5 usb_string_descriptor_struct_product Struct Reference	66
4.5.1 Member Data Documentation	66
4.5.1.1 bDescriptorType	66
4.5.1.2 bLength	66
4.5.1.3 wString	66
4.6 usb_string_descriptor_struct_serial_number Struct Reference	66
4.6.1 Member Data Documentation	66
4.6.1.1 bDescriptorType	66
4.6.1.2 bLength	66
4.6.1.3 wString	66
<b>5 File Documentation</b>	<b>67</b>
5.1 parselib2.cpp File Reference	67
5.1.1 Function Documentation	68
5.1.1.1 basenamef()	68
5.1.1.2 countWords()	69
5.1.1.3 eos()	69
5.1.1.4 eow()	69
5.1.1.5 nextWord()	70
5.1.1.6 scaling()	70
5.1.1.7 serialPrintf()	71
5.1.1.8 serialPrintInf()	71
5.1.1.9 strBool()	71
5.1.1.10 strFlt()	72
5.1.1.11 strFlts()	73
5.1.1.12 strMatch()	73
5.1.1.13 strUInt()	74
5.1.1.14 strUInt16()	74
5.1.1.15 strUInt32()	75
5.1.1.16 strUInt32s()	75
5.1.1.17 strUInt8()	76
5.1.1.18 strUInt8lim()	76
5.1.1.19 strUInts()	77
5.1.1.20 wordLength()	77
5.1.2 Variable Documentation	77

5.1.2.1 nprintbuffer . . . . .	77
5.1.2.2 printbuffer . . . . .	78
5.2 parselib2.h File Reference . . . . .	78
5.2.1 Macro Definition Documentation . . . . .	79
5.2.1.1 PARSELIB_H . . . . .	79
5.2.2 Function Documentation . . . . .	79
5.2.2.1 basenamef() . . . . .	79
5.2.2.2 countWords() . . . . .	79
5.2.2.3 nextWord() . . . . .	80
5.2.2.4 serialPrintf() . . . . .	80
5.2.2.5 serialPrintInf() . . . . .	81
5.2.2.6 strBool() . . . . .	81
5.2.2.7 strFlt() . . . . .	81
5.2.2.8 strFlts() . . . . .	82
5.2.2.9 strMatch() . . . . .	83
5.2.2.10 strUInt() . . . . .	83
5.2.2.11 strUInt16() . . . . .	84
5.2.2.12 strUInt32() . . . . .	84
5.2.2.13 strUInt32s() . . . . .	85
5.2.2.14 strUInt8() . . . . .	85
5.2.2.15 strUInt8lim() . . . . .	86
5.2.2.16 strUInts() . . . . .	86
5.2.2.17 wordLength() . . . . .	87
5.3 parselib2.h . . . . .	87
5.4 README.md File Reference . . . . .	88
5.5 TCD1304Device2.h File Reference . . . . .	88
5.5.1 Macro Definition Documentation . . . . .	91
5.5.1.1 BUSY_PIN . . . . .	91
5.5.1.2 BUSY_PIN_DEFAULT . . . . .	91
5.5.1.3 CLEARCNVST [1/2] . . . . .	91
5.5.1.4 CLEARCNVST [2/2] . . . . .	91
5.5.1.5 CLK_CHANNEL . . . . .	91
5.5.1.6 CLK_CMPF_MASK . . . . .	91
5.5.1.7 CLK_CTRL2_MASK . . . . .	92
5.5.1.8 CLK_DEFAULT . . . . .	92
5.5.1.9 CLK_IRQ . . . . .	92
5.5.1.10 CLK_MASK . . . . .	92
5.5.1.11 CLK_MONITOR_PIN . . . . .	92
5.5.1.12 CLK_MUXVAL . . . . .	92

---

5.5.1.13 CLK_PIN . . . . .	92
5.5.1.14 CLK_SUBMODULE . . . . .	92
5.5.1.15 CMPF_MASKA_OFF . . . . .	92
5.5.1.16 CMPF_MASKA_ON . . . . .	92
5.5.1.17 CMPF_MASKA_ON_OFF . . . . .	93
5.5.1.18 CMPF_MASKB_OFF . . . . .	93
5.5.1.19 CMPF_MASKB_ON . . . . .	93
5.5.1.20 CMPF_MASKB_ON_OFF . . . . .	93
5.5.1.21 CNVST_CHANNEL . . . . .	93
5.5.1.22 CNVST_CMPF_MASK . . . . .	93
5.5.1.23 CNVST_CTRL2_MASK . . . . .	93
5.5.1.24 CNVST_IRQ . . . . .	93
5.5.1.25 CNVST_MASK . . . . .	93
5.5.1.26 CNVST_PIN . . . . .	93
5.5.1.27 CNVST_PULSE_SECS . . . . .	94
5.5.1.28 CNVST_SUBMODULE . . . . .	94
5.5.1.29 COUNTER_MAX_SECS . . . . .	94
5.5.1.30 CYCCNT2SECS . . . . .	94
5.5.1.31 DATASTART . . . . .	94
5.5.1.32 DATASTOP . . . . .	94
5.5.1.33 DEBUGPRINTF . . . . .	94
5.5.1.34 ICG_CHANNEL . . . . .	94
5.5.1.35 ICG_CMPF_MASK . . . . .	94
5.5.1.36 ICG_CTRL2_MASK . . . . .	95
5.5.1.37 ICG_IRQ . . . . .	95
5.5.1.38 ICG_MASK . . . . .	95
5.5.1.39 ICG_MUXVAL . . . . .	95
5.5.1.40 ICG_PIN . . . . .	95
5.5.1.41 ICG_SUBMODULE . . . . .	95
5.5.1.42 NBITS . . . . .	95
5.5.1.43 NBYTES . . . . .	95
5.5.1.44 NBYTES32 . . . . .	95
5.5.1.45 NDARK . . . . .	95
5.5.1.46 NPIXELS . . . . .	96
5.5.1.47 NREADOUT . . . . .	96
5.5.1.48 PINPULLS . . . . .	96
5.5.1.49 PWM_CTRL2_CLOCK_MASTER . . . . .	96
5.5.1.50 PWM_CTRL2_CLOCK_SLAVE . . . . .	96
5.5.1.51 PWM_CTRL2_CLOCK_SYNC . . . . .	96

5.5.1.52 ROUNDTO . . . . .	96
5.5.1.53 ROUNDTOMOD . . . . .	96
5.5.1.54 ROUNDUP . . . . .	97
5.5.1.55 SETCNVST [1/2] . . . . .	97
5.5.1.56 SETCNVST [2/2] . . . . .	97
5.5.1.57 SH_CHANNEL . . . . .	97
5.5.1.58 SH_CLEARING_DEFAULT . . . . .	97
5.5.1.59 SH_CMPF_MASK . . . . .	97
5.5.1.60 SH_CTRL2_MASK . . . . .	97
5.5.1.61 SH_IRQ . . . . .	97
5.5.1.62 SH_MASK . . . . .	97
5.5.1.63 SH_MUXVAL . . . . .	98
5.5.1.64 SH_PIN . . . . .	98
5.5.1.65 SH_STOP_IN_READ . . . . .	98
5.5.1.66 SH_SUBMODULE . . . . .	98
5.5.1.67 SHUTTERMIN . . . . .	98
5.5.1.68 SYNC_PIN . . . . .	98
5.5.1.69 SYNC_PIN_DEFAULT . . . . .	98
5.5.1.70 TCD1304_MAXCLKHZ . . . . .	98
5.5.1.71 TCD1304_MINCLKHZ . . . . .	98
5.5.1.72 TDIFF . . . . .	99
5.5.1.73 TIMER_CHANNEL . . . . .	99
5.5.1.74 TIMER_CMPF_MASK . . . . .	99
5.5.1.75 TIMER_CTRL2_MASK . . . . .	99
5.5.1.76 TIMER_IRQ . . . . .	99
5.5.1.77 TIMER_MASK . . . . .	99
5.5.1.78 TIMER_MUXVAL . . . . .	99
5.5.1.79 TIMER_PIN . . . . .	99
5.5.1.80 TIMER_SUBMODULE . . . . .	99
5.5.1.81 TRIGGER_PIN . . . . .	100
5.5.1.82 USBSPEED . . . . .	100
5.5.1.83 USBTRANSFERSECS . . . . .	100
5.5.1.84 VFS . . . . .	100
5.5.1.85 VPERBIT . . . . .	100
5.5.2 Enumeration Type Documentation . . . . .	100
5.5.2.1 TCD1304_Mode_t . . . . .	100
5.6 TCD1304Device2.h . . . . .	101
5.7 TCD1304Device_Controller_251208.ino File Reference . . . . .	140
5.7.1 Macro Definition Documentation . . . . .	143

5.7.1.1 ADC_RESOLUTION . . . . .	143
5.7.1.2 ASCII . . . . .	143
5.7.1.3 BINARY . . . . .	144
5.7.1.4 CONTROLLER_CSPIN . . . . .	144
5.7.1.5 CONTROLLER_CSPIN2 . . . . .	144
5.7.1.6 CONTROLLER_OCPIN . . . . .	144
5.7.1.7 CYCLES_PER_USEC . . . . .	144
5.7.1.8 DRMCNELSONLAB . . . . .	144
5.7.1.9 EEPROM_COEFF_ADDR . . . . .	144
5.7.1.10 EEPROM_COEFF_LEN . . . . .	144
5.7.1.11 EEPROM_ID_ADDR . . . . .	144
5.7.1.12 EEPROM_ID_LEN . . . . .	144
5.7.1.13 EEPROM_NCOEFFS . . . . .	145
5.7.1.14 EEPROM_NUNITS . . . . .	145
5.7.1.15 EEPROM_SIZE . . . . .	145
5.7.1.16 EEPROM_UNITS_ADDR . . . . .	145
5.7.1.17 EEPROM_UNITS_LEN . . . . .	145
5.7.1.18 IMR_INDEX . . . . .	145
5.7.1.19 ISR_INDEX . . . . .	145
5.7.1.20 NADC_CHANNELS . . . . .	145
5.7.1.21 NBUFFERS . . . . .	145
5.7.1.22 RCVLEN . . . . .	145
5.7.1.23 SNDLEN . . . . .	146
5.7.1.24 thisMANUFACTURER_NAME . . . . .	146
5.7.1.25 thisMANUFACTURER_NAME_LEN . . . . .	146
5.7.1.26 thisPRODUCT_NAME . . . . .	146
5.7.1.27 thisPRODUCT_NAME_LEN . . . . .	146
5.7.1.28 thisPRODUCT_SERIAL_NUMBER . . . . .	146
5.7.1.29 thisPRODUCT_SERIAL_NUMBER_LEN . . . . .	146
5.7.2 Function Documentation . . . . .	146
5.7.2.1 blink() . . . . .	146
5.7.2.2 calculateCRC() . . . . .	147
5.7.2.3 clock_isr() . . . . .	147
5.7.2.4 clock_setup() . . . . .	147
5.7.2.5 clock_start() . . . . .	148
5.7.2.6 clock_stop() . . . . .	148
5.7.2.7 cycles64() . . . . .	149
5.7.2.8 disablePinInterrupts() . . . . .	149
5.7.2.9 eeErase() . . . . .	149

5.7.2.10 eeread()	149
5.7.2.11 eereadUntil()	150
5.7.2.12 eewrite()	150
5.7.2.13 eewriteUntil()	150
5.7.2.14 eraseCoefficients()	151
5.7.2.15 eraseIdentifier()	152
5.7.2.16 eraseUnits()	152
5.7.2.17 fastAnalogRead()	153
5.7.2.18 frame_callback()	153
5.7.2.19 help()	154
5.7.2.20 loop()	154
5.7.2.21 ocpinAttach()	155
5.7.2.22 ocpinDetach()	156
5.7.2.23 ocpinISR()	156
5.7.2.24 ocpinRead()	157
5.7.2.25 printCoefficients()	157
5.7.2.26 printIdentifier()	158
5.7.2.27 printTriggerSetup()	158
5.7.2.28 printUnits()	159
5.7.2.29 pulsePin()	159
5.7.2.30 readCoefficients()	160
5.7.2.31 readIdentifier()	160
5.7.2.32 readUnits()	161
5.7.2.33 resumePinInterrupts()	162
5.7.2.34 send_frame()	162
5.7.2.35 send_frames()	162
5.7.2.36 send_header()	163
5.7.2.37 sendADCs()	163
5.7.2.38 sendBuffer_Binary()	164
5.7.2.39 sendBuffer_Formatted()	164
5.7.2.40 sendChipTemperature()	165
5.7.2.41 sendData()	165
5.7.2.42 sendDataCRC()	166
5.7.2.43 sendDataReady()	166
5.7.2.44 sendDataSum()	167
5.7.2.45 sendTestData()	167
5.7.2.46 setup()	168
5.7.2.47 spi_settings()	168
5.7.2.48 storeCoefficients()	169

---

5.7.2.49 storeIdentifier()	169
5.7.2.50 storeUnits()	170
5.7.2.51 strPin()	170
5.7.2.52 strSubModule()	171
5.7.2.53 strTrigger()	172
5.7.2.54 sumData()	173
5.7.2.55 tempmonGetTemp()	174
5.7.3 Variable Documentation	174
5.7.3.1 adc	174
5.7.3.2 adc_averages	174
5.7.3.3 adc_data	174
5.7.3.4 analogPin	174
5.7.3.5 authorstr	174
5.7.3.6 buffer_index	175
5.7.3.7 buffer_ring	175
5.7.3.8 bufferp	175
5.7.3.9 busyPin	175
5.7.3.10 chipTemp_averages	175
5.7.3.11 CLKPin	175
5.7.3.12 clock_callback	175
5.7.3.13 clock_counter	175
5.7.3.14 clock_counts	175
5.7.3.15 clock_mode	175
5.7.3.16 clock_timer	176
5.7.3.17 clock_usecs	176
5.7.3.18 CNVSTPin	176
5.7.3.19 counter	176
5.7.3.20 crc_enable	176
5.7.3.21 crctable	176
5.7.3.22 data_async	176
5.7.3.23 dataformat	177
5.7.3.24 diagnostic_usecs	177
5.7.3.25 diagnostics	177
5.7.3.26 elapsed_usecs	177
5.7.3.27 fMPin	177
5.7.3.28 frame_header_ring	177
5.7.3.29 frame_index	177
5.7.3.30 frame_send_index	177
5.7.3.31 framep	177

5.7.3.32 ICGPin . . . . .	177
5.7.3.33 nrcvbuf . . . . .	178
5.7.3.34 ocpinattached . . . . .	178
5.7.3.35 ocpinstate . . . . .	178
5.7.3.36 rcvbuffer . . . . .	178
5.7.3.37 SDIPin . . . . .	178
5.7.3.38 SDOPin . . . . .	178
5.7.3.39 sensorstr . . . . .	178
5.7.3.40 SHPin . . . . .	178
5.7.3.41 sndbuffer . . . . .	178
5.7.3.42 srcfilestr . . . . .	178
5.7.3.43 sum_enable . . . . .	179
5.7.3.44 syncPin . . . . .	179
5.7.3.45 tcd1304device . . . . .	179
5.7.3.46 triggerPin . . . . .	179
5.7.3.47 usb_string_manufacturer_name . . . . .	179
5.7.3.48 usb_string_product_name . . . . .	179
5.7.3.49 usb_string_serial_number . . . . .	179
5.7.3.50 versionstr . . . . .	179

<b>Index</b>	<b>181</b>
--------------	------------



# Chapter 1

## Firmware for the TCD1304 using FlexPWM

This directory contains an Arduino sketch file and libraries to operate the TCD1304 Sensor using the Teensy 4 in either the two board system, the TCD1304 SPI board plus Teensy 4 Controller Rev 3 or the All-In-One board with FlexPWM support.

In this version we add ring buffering. Regard this as experimental for the time being. In preliminary testing with an Intel I5 host running Linux, it seems to work well. We want to test more extensively on a wider range of host platforms.

To setup your system to install the firmware, see the [Teensyduino website and setup instructions](#)

See the [Python directory in this repo](#) for an interactive user utility with real time graphics. The Python program uses Python3, Numpy, Matplotlib, PySerial and SciPY.

In the interactive user utility, use the command "help" to see a list of commands.

### 1.0.1 In this directory:

The file [TCD1304Device2.h](#) is a header only library for the TCD1304 using the NXP iMXRT1060 FlexPWM.

The file [TCD1304Device\\_Controller\\_251208.ino](#) implements the user command interface and instantiates the ring buffer system.

The files [parselib2.cpp](#) and [parselib2.h](#) provides an api for parsing commands and is used in the controller code to implement the command language for the device.

The ring buffer system is implemented in the controller code, using the frame completion callback interface provide by the [TCD1304Device](#) class along with a frame structure also provided in the device library.



## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">TCD1304Device::Frame_Header_struct</a>	7
<a href="#">TCD1304Device::SubModule</a>	10
<a href="#">TCD1304Device</a>	14
<a href="#">usb_string_descriptor_struct_manufacturer</a>	65
<a href="#">usb_string_descriptor_struct_product</a>	66
<a href="#">usb_string_descriptor_struct_serial_number</a>	66



# Chapter 3

## File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">parselib2.cpp</a>	67
<a href="#">parselib2.h</a>	78
<a href="#">TCD1304Device2.h</a>	88
<a href="#">TCD1304Device_Controller_251208.ino</a>	140



# Chapter 4

## Class Documentation

### 4.1 TCD1304Device::Frame\_Header\_struct Struct Reference

```
#include <TCD1304Device2.h>
```

#### Public Attributes

- uint16\_t \* [buffer](#)
- unsigned int [nbuffer](#)
- unsigned int [avgdummy](#)
- float [offset](#)
- float [frame\\_elapsed\\_secs](#)
- float [frame\\_exposure\\_secs](#)
- float [timer\\_elapsed\\_secs](#)
- float [timer\\_difference\\_secs](#)
- float [trigger\\_elapsed\\_secs](#)
- float [trigger\\_difference\\_secs](#)
- unsigned int [frame\\_counter](#)
- unsigned int [frameset\\_counter](#)
- unsigned int [trigger\\_counter](#)
- [TCD1304\\_Mode\\_t](#) [mode](#)
- bool [trigger\\_mode](#)
- bool [error\\_flag](#)
- bool [oops\\_flag](#)
- bool [frames\\_completed](#)
- bool [framesets\\_completed](#)
- bool [ready\\_for\\_send](#)

#### 4.1.1 Member Data Documentation

##### 4.1.1.1 [avgdummy](#)

```
unsigned int TCD1304Device::Frame_Header_struct::avgdummy
```

#### 4.1.1.2 buffer

`uint16_t* TCD1304Device::Frame_Header_struct::buffer`

#### 4.1.1.3 error\_flag

`bool TCD1304Device::Frame_Header_struct::error_flag`

#### 4.1.1.4 frame\_counter

`unsigned int TCD1304Device::Frame_Header_struct::frame_counter`

#### 4.1.1.5 frame\_elapsed\_secs

`float TCD1304Device::Frame_Header_struct::frame_elapsed_secs`

#### 4.1.1.6 frame\_exposure\_secs

`float TCD1304Device::Frame_Header_struct::frame_exposure_secs`

#### 4.1.1.7 frames\_completed

`bool TCD1304Device::Frame_Header_struct::frames_completed`

#### 4.1.1.8 frameset\_counter

`unsigned int TCD1304Device::Frame_Header_struct::frameset_counter`

#### 4.1.1.9 framesets\_completed

`bool TCD1304Device::Frame_Header_struct::framesets_completed`

#### 4.1.1.10 mode

`TCD1304\_Mode\_t TCD1304Device::Frame_Header_struct::mode`

#### 4.1.1.11 nbuffer

`unsigned int TCD1304Device::Frame_Header_struct::nbuffer`



#### 4.1.1.12 offset

```
float TCD1304Device::Frame_Header_struct::offset
```

#### 4.1.1.13 oops\_flag

```
bool TCD1304Device::Frame_Header_struct::oops_flag
```

#### 4.1.1.14 ready\_for\_send

```
bool TCD1304Device::Frame_Header_struct::ready_for_send
```

#### 4.1.1.15 timer\_difference\_secs

```
float TCD1304Device::Frame_Header_struct::timer_difference_secs
```

#### 4.1.1.16 timer\_elapsed\_secs

```
float TCD1304Device::Frame_Header_struct::timer_elapsed_secs
```

#### 4.1.1.17 trigger\_counter

```
unsigned int TCD1304Device::Frame_Header_struct::trigger_counter
```

#### 4.1.1.18 trigger\_difference\_secs

```
float TCD1304Device::Frame_Header_struct::trigger_difference_secs
```

#### 4.1.1.19 trigger\_elapsed\_secs

```
float TCD1304Device::Frame_Header_struct::trigger_elapsed_secs
```

#### 4.1.1.20 trigger\_mode

```
bool TCD1304Device::Frame_Header_struct::trigger_mode
```

The documentation for this struct was generated from the following file:

- [TCD1304Device2.h](#)

## 4.2 TCD1304Device::SubModule Struct Reference

```
#include <TCD1304Device2.h>
```

### Public Member Functions

- [SubModule](#) (const char \*name\_, uint8\_t submod\_, uint8\_t mask\_, uint8\_t pinA\_, uint8\_t muxvalA\_, uint8\_t pinB\_, uint8\_t muxvalB\_, IRQ\_NUMBER\_t irq\_, IMXRT\_FLEXPWM\_t \*flexpwm\_)

### Public Attributes

- const char \* [name](#)
- const uint8\_t [submod](#)
- const uint8\_t [mask](#)
- const uint8\_t [pinA](#)
- const uint8\_t [muxvalA](#)
- const uint8\_t [pinB](#)
- const uint8\_t [muxvalB](#)
- IRQ\_NUMBER\_t [irq](#)
- IMXRT\_FLEXPWM\_t \* [flexpwm](#)
- uint16\_t [period\\_counts](#) = 0
- uint16\_t [onA\\_counts](#) = 0
- uint16\_t [offA\\_counts](#) = 0
- uint16\_t [onB\\_counts](#) = 0
- uint16\_t [offB\\_counts](#) = 0
- uint16\_t [ctrl2\\_mask](#) = 0
- uint16\_t [intena\\_mask](#) = 0
- uint16\_t [divider](#) = 1
- uint8\_t [prescale](#) = 0
- uint8\_t [filler](#) = 0
- float [period\\_secs](#) = 0
- bool [invertA](#) = false
- bool [invertB](#) = false
- bool [newvals](#) = false
- void(\* [isr](#) )() = nullptr
- uint16\_t [inten\\_mask](#) = 0

### 4.2.1 Constructor & Destructor Documentation

#### 4.2.1.1 SubModule()

```
TCD1304Device::SubModule::SubModule (
    const char * name_,
    uint8_t submod_,
    uint8_t mask_,
    uint8_t pinA_,
    uint8_t muxvalA_,
    uint8_t pinB_,
    uint8_t muxvalB_,
    IRQ_NUMBER_t irq_,
    IMXRT_FLEXPWM_t * flexpwm_) [inline]
```

## 4.2.2 Member Data Documentation

### 4.2.2.1 ctrl2\_mask

```
uint16_t TCD1304Device::SubModule::ctrl2_mask = 0
```

### 4.2.2.2 divider

```
uint16_t TCD1304Device::SubModule::divider = 1
```

### 4.2.2.3 filler

```
uint8_t TCD1304Device::SubModule::filler = 0
```

### 4.2.2.4 flexpwm

```
IMXRT_FLEXPWM_t* TCD1304Device::SubModule::flexpwm
```

### 4.2.2.5 inten\_mask

```
uint16_t TCD1304Device::SubModule::inten_mask = 0
```

### 4.2.2.6 intena\_mask

```
uint16_t TCD1304Device::SubModule::intena_mask = 0
```

### 4.2.2.7 invertA

```
bool TCD1304Device::SubModule::invertA = false
```

### 4.2.2.8 invertB

```
bool TCD1304Device::SubModule::invertB = false
```

### 4.2.2.9 irq

```
IRQ_NUMBER_t TCD1304Device::SubModule::irq
```

**4.2.2.10 isr**

```
void(* TCD1304Device::SubModule::isr) () = nullptr
```

**4.2.2.11 mask**

```
const uint8_t TCD1304Device::SubModule::mask
```

**4.2.2.12 muxvalA**

```
const uint8_t TCD1304Device::SubModule::muxvalA
```

**4.2.2.13 muxvalB**

```
const uint8_t TCD1304Device::SubModule::muxvalB
```

**4.2.2.14 name**

```
const char* TCD1304Device::SubModule::name
```

**4.2.2.15 newvals**

```
bool TCD1304Device::SubModule::newvals = false
```

**4.2.2.16 offA\_counts**

```
uint16_t TCD1304Device::SubModule::offA_counts = 0
```

**4.2.2.17 offB\_counts**

```
uint16_t TCD1304Device::SubModule::offB_counts = 0
```

**4.2.2.18 onA\_counts**

```
uint16_t TCD1304Device::SubModule::onA_counts = 0
```

**4.2.2.19 onB\_counts**

```
uint16_t TCD1304Device::SubModule::onB_counts = 0
```

#### 4.2.2.20 period\_counts

```
uint16_t TCD1304Device::SubModule::period_counts = 0
```

#### 4.2.2.21 period\_secs

```
float TCD1304Device::SubModule::period_secs = 0
```

#### 4.2.2.22 pinA

```
const uint8_t TCD1304Device::SubModule::pinA
```

#### 4.2.2.23 pinB

```
const uint8_t TCD1304Device::SubModule::pinB
```

#### 4.2.2.24 prescale

```
uint8_t TCD1304Device::SubModule::prescale = 0
```

#### 4.2.2.25 submod

```
const uint8_t TCD1304Device::SubModule::submod
```

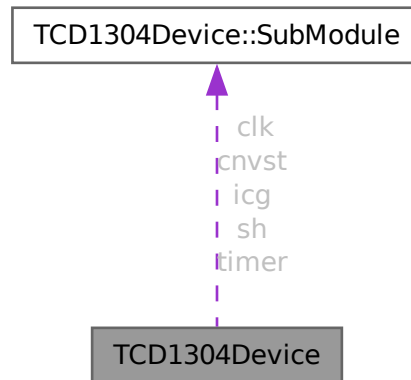
The documentation for this struct was generated from the following file:

- [TCD1304Device2.h](#)

## 4.3 TCD1304Device Class Reference

```
#include <TCD1304Device2.h>
```

Collaboration diagram for TCD1304Device:



### Classes

- struct [Frame\\_Header\\_struct](#)
- struct [SubModule](#)

### Public Types

- typedef struct [TCD1304Device::Frame\\_Header\\_struct](#) [Frame\\_Header](#)

### Public Member Functions

- [TCD1304Device](#) (unsigned int period=[CLK\\_DEFAULT](#))
- void [stop\\_all](#) ()
- bool [read](#) (uint nframes, float exposure, uint16\_t \*[bufferp](#), void(\*frame\_callback)(), void(\*frameset\_callback)(), void(\*completion\_callback)(), void(\*setup\_callback)(), bool start=true)
- bool [read](#) (uint nframes, float exposure, float frame\_interval, uint16\_t \*[bufferp](#), void(\*frame\_callback)(), void(\*frameset\_callback)(), void(\*completion\_callback)(), void(\*setup\_callback)(), bool start=true)
- bool [start\\_read](#) ()
- bool [wait\\_read](#) (float timeout=0., float timestep=0.01, bool verbose=false)
- bool [triggered\\_read](#) (uint ntriggers, uint nframes, float exposure, uint16\_t \*[bufferp](#), void(\*frame\_callback)(), void(\*frameset\_callback)(), void(\*completion\_callback)(), void(\*setup\_callback)(), bool start=true)
- bool [triggered\\_read](#) (uint ntriggers, uint nframes, float exposure, float interval, uint16\_t \*[bufferp](#), void(\*frame\_callback)(), void(\*frameset\_callback)(), void(\*completion\_callback)(), void(\*setup\_callback)(), bool start=true)

- bool [wait\\_triggered\\_read](#) (float timeout=1., float timestep=0.01, bool verbose=false)
- void [print\\_submodule](#) (SubModule \*p)
- bool [check\\_submodule](#) (SubModule \*p)
- bool [print\\_and\\_check\\_submodule](#) (SubModule \*p)
- void [load\\_submodule](#) (SubModule \*p)
- bool [setup\\_submodule](#) (SubModule \*p, uint8\_t prescale, uint16\_t period\_counts, uint16\_t onA\_counts, uint16\_t offA\_counts, bool invertA, uint16\_t onB\_counts, uint16\_t offB\_counts, bool invertB, uint16\_t ctrl2\_mask)
- void [attach\\_isr](#) (SubModule \*p, uint16\_t cmpf\_mask, void(\*isrf)())
- void [clear\\_frames\\_completed\\_callback](#) ()
- void [load\\_frames\\_completed\\_callback](#) (void(\*callback)(), unsigned int nframes=0)
- void [clear\\_framesets\\_completed\\_callback](#) ()
- void [load\\_framesets\\_completed\\_callback](#) (void(\*callback)(), unsigned int nsets=0)
- bool [flexpwm\\_wait](#) (float timeout\_=1., float timestep\_=0.01, bool verbose=false)
- bool [setup\\_pulse](#) (float clk\_secs, float sh\_secs, float sh\_offset\_secs, float icg\_secs, float icg\_offset\_secs, uint16\_t \*buffer, unsigned int nbuffer, void(\*callback)())
- bool [setup\\_frameset](#) (float clk\_secs, float sh\_secs, float sh\_offset\_secs, float icg\_secs, float icg\_offset\_secs, float exposure\_secs, float frame\_interval\_secs, unsigned int nframes, uint16\_t \*buffer, unsigned int nbuffer, void(\*callback)())
- bool [timer\\_wait](#) (float timeout\_=1., float timestep\_=0.01, bool verbose=false)
- bool [setup\\_timer](#) (float exposure\_secs, float exposure\_offset\_secs, unsigned int ncounts=0)
- bool [start\\_triggers](#) ()
- bool [wait\\_triggers](#) (float timeout\_=1., float timestep\_=0.01, bool verbose=false)
- bool [setup\\_triggers](#) (unsigned int ncounts=0)
- void [set\\_clock\\_master](#) (uint8\_t submod, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))

*Low level functions for command line register access.*

- void [set\\_clock\\_master](#) (SubModule \*p)
- void [set\\_clock\\_slave](#) (uint8\_t submod, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [set\\_clock\\_slave](#) (SubModule \*p)
- void [set\\_clock\\_sync](#) (uint8\_t submod, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [set\\_clock\\_sync](#) (SubModule \*p)
- void [clear\\_ldok](#) (uint8\_t mask, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [set\\_ldok](#) (uint8\_t mask, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [clear\\_run](#) (uint8\_t mask, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [set\\_run](#) (uint8\_t mask, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [force](#) (uint8\_t submod, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [set\\_prescale](#) (uint8\_t submod, uint8\_t prescale=0, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [set\\_init](#) (uint8\_t submod, uint16\_t value, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [set\\_val0](#) (uint8\_t submod, uint16\_t value, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [set\\_val1](#) (uint8\_t submod, uint16\_t value, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [set\\_val2](#) (uint8\_t submod, uint16\_t value, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [set\\_val3](#) (uint8\_t submod, uint16\_t value, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [set\\_val4](#) (uint8\_t submod, uint16\_t value, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [set\\_val5](#) (uint8\_t submod, uint16\_t value, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- uint16\_t [set\\_outen](#) (uint16\_t mask16, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- uint16\_t [set\\_outen\\_on](#) (uint16\_t mask16, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- uint16\_t [set\\_outen\\_off](#) (uint16\_t mask16, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- uint16\_t [set\\_outenA\\_on](#) (uint8\_t mask8, IMXRT\_FLEXPWM\_t \*q=[flexpwm](#))
- void [printbits16\\_](#) (uint16\_t u16, int bits)
- void [register\\_dump](#) (IMXRT\_FLEXPWM\_t \*const q=[flexpwm](#), uint8\_t mask=0xFF)

## Static Public Member Functions

- static void [stop\\_runs\\_only](#) ()
- static void [disable\\_irqs](#) ()
- static void [stop\\_with\\_irqs](#) ()
- static void [clear\\_error\\_flags](#) ()
- static void [clear\\_mode](#) ()
- static void [clear\\_sync\\_busy\\_pins](#) ()
- static void [toggle\\_busypin](#) ()
- static void [toggle\\_syncpin](#) ()
- static void [clear\\_busypin](#) ()
- static void [clear\\_syncpin](#) ()
- static void [update\\_read\\_buffer](#) (uint16\_t \*buffer)
- static void [fill\\_frame\\_header](#) (Frame\_Header \*p)
- static uint64\_t [cycles64](#) ()
- static double [sh\\_elapsed\\_secs](#) ()
- static double [sh\\_difference\\_secs](#) ()
- static double [sh\\_exposure\\_secs](#) ()
- static double [timer\\_elapsed\\_secs](#) ()
- static double [timer\\_difference\\_secs](#) ()
- static double [trigger\\_elapsed\\_secs](#) ()
- static double [trigger\\_difference\\_secs](#) ()
- static void [print\\_errormsg](#) (const char \*name, const char \*errmsg)
- static void [print\\_errormsg](#) (SubModule \*p, const char \*errmsg)
- static void [print\\_counters](#) ()
- static void [flexpwm\\_start](#) ()
- static void [flexpwm\\_stop](#) ()
- static void [pulse\\_sh\\_isr](#) ()
- static void [pulse\\_icg\\_isr](#) ()
- static void [pulse\\_cnvst\\_isr](#) ()
- static void [pulse\\_start](#) ()
- static void [pulse\\_arm](#) ()
- static void [pulse\\_init\\_frames](#) ()
- static void [pulse\\_init\\_frameset](#) ()
- static void [frameset\\_sh\\_isr](#) ()
- static void [frameset\\_icg\\_isr](#) ()
- static void [frameset\\_cnvst\\_isr](#) ()
- static void [frameset\\_start](#) ()
- static void [frameset\\_arm](#) ()
- static void [frameset\\_init\\_frames](#) ()
- static void [frameset\\_init\\_frameset](#) ()
- static void [timer\\_isr](#) ()
- static void [timer\\_start](#) ()
- static void [timer\\_stop](#) ()
- static void [timer\\_stop\\_with\\_irq](#) ()
- static void [trigger\\_isr](#) ()
- static void [stop\\_triggers](#) ()
- static void [setup\\_digital\\_pins](#) ()
- static void [close](#) ()



### Static Public Attributes

- static uint64\_t [sh\\_cycCnt64\\_start](#) = 0
- static uint64\_t [sh\\_cycCnt64\\_now](#) = 0
- static uint64\_t [sh\\_cycCnt64\\_prev](#) = 0
- static uint64\_t [sh\\_cycCnt64\\_exposure](#) = 0
- static uint64\_t [timer\\_cycCnt64\\_start](#) = 0
- static uint64\_t [timer\\_cycCnt64\\_now](#) = 0
- static uint64\_t [timer\\_cycCnt64\\_prev](#) = 0
- static uint64\_t [trigger\\_cycCnt64\\_start](#) = 0
- static uint64\_t [trigger\\_cycCnt64\\_now](#) = 0
- static uint64\_t [trigger\\_cycCnt64\\_prev](#) = 0
- static [TCD1304\\_Mode\\_t](#) [mode](#) = NOTCONFIGURED
- static bool [trigger\\_mode](#) = false
- static bool [trigger\\_attached](#) = false
- static unsigned int [sh\\_counter](#) = 0
- static unsigned int [icg\\_counter](#) = 0
- static unsigned int [cnvst\\_counter](#) = 0
- static unsigned int [sh\\_counts\\_per\\_icg](#) = 0
- static unsigned int [sh\\_clearing\\_counts](#) = SH\_CLEARING\_DEFAULT
- static unsigned int [sh\\_clearing\\_counter](#) = 0
- static unsigned int [sh\\_short\\_period\\_counts](#) = 0
- static unsigned int [read\\_counter](#) = 0
- static unsigned int [read\\_counts](#) = 0
- static uint16\_t \* [read\\_buffer](#) = 0
- static uint16\_t \* [read\\_pointer](#) = 0
- static void(\* [read\\_callback](#) )()=0
- static unsigned int [frame\\_counter](#) = 0
- static unsigned int [frame\\_counts](#) = 0
- static void(\* [frames\\_completed\\_callback](#) )()=0
- static unsigned int [frameset\\_counter](#) = 0
- static unsigned int [frameset\\_counts](#) = 0
- static void(\* [framesets\\_completed\\_callback](#) )()=0
- static unsigned int [timer\\_inner\\_counter](#) = 0
- static unsigned int [timer\\_inner\\_counts](#) = 0
- static unsigned int [timer\\_outer\\_counter](#) = 0
- static unsigned int [timer\\_outer\\_counts](#) = 0
- static void(\* [timer\\_callback](#) )()=0
- static float [timer\\_interframe\\_min\\_secs](#) = 0.
- static float [timer\\_period\\_secs](#) = 0
- static float [timer\\_interval\\_secs](#) = .0
- static unsigned int [trigger\\_counter](#) = 0
- static unsigned int [trigger\\_counts](#) = 1
- static void(\* [trigger\\_callback](#) )()=0
- static uint8\_t [trigger\\_pin](#) = TRIGGER\_PIN
- static uint8\_t [trigger\\_edge\\_mode](#) = RISING
- static uint8\_t [trigger\\_pin\\_mode](#) = INPUT
- static uint16\_t [cnvst\\_extra\\_delay\\_counts](#) = 1
- static bool [skip\\_one](#) = false
- static bool [skip\\_one\\_reload](#) = false
- static bool [busytoggled](#) = false

- static uint8\_t `sync_pin` = `SYNC_PIN`
- static bool `synctoggled` = false
- static bool `sync_enabled` = true
- static IMXRT\_FLEXPWM\_t \*const `flexpwm` = &IMXRT\_FLEXPWM2
- static SubModule `clk` = {"clk", CLK\_SUBMODULE, CLK\_MASK, CLK\_PIN, CLK\_MUXVAL, 0xFF, 0, CLK\_IRQ, &IMXRT\_FLEXPWM2}
- static SubModule `sh` = {"sh", SH\_SUBMODULE, SH\_MASK, SH\_PIN, SH\_MUXVAL, 0xFF, 0, SH\_IRQ, &IMXRT\_FLEXPWM2}
- static SubModule `icg` = {"icg", ICG\_SUBMODULE, ICG\_MASK, ICG\_PIN, ICG\_MUXVAL, 0xFF, 0, ICG\_IRQ, &IMXRT\_FLEXPWM2}
- static SubModule `cnvst` = {"cnvst", CNVST\_SUBMODULE, CNVST\_MASK, 0xFF, 0, 0xFF, 0, CNVST\_IRQ, &IMXRT\_FLEXPWM2}
- static IMXRT\_FLEXPWM\_t \*const `timerflexpwm` = &IMXRT\_FLEXPWM4
- static SubModule `timer` = {"timer", TIMER\_SUBMODULE, TIMER\_MASK, 0xFF, 0, TIMER\_PIN, TIMER\_MUXVAL, TIMER\_IRQ, &IMXRT\_FLEXPWM4}
- static bool `error_flag` = false
- static bool `oops_flag` = false
- static float `read_expected_time` = 0.
- static bool `flexpwm_running` = false
- static bool `pulse_armed` = false

*Configure the flexpwm for single pulse, use this with a timer.*

- static bool `frameset_armed` = false
- static bool `timer_first_time_flag` = true
- static bool `timer_running` = false
- static bool `trigger_busy` = false

## 4.3.1 Member Typedef Documentation

### 4.3.1.1 Frame\_Header

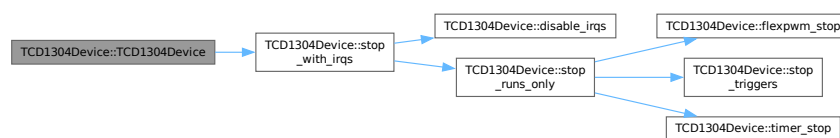
```
typedef struct TCD1304Device::Frame_Header_struct TCD1304Device::Frame_Header
```

## 4.3.2 Constructor & Destructor Documentation

### 4.3.2.1 TCD1304Device()

```
TCD1304Device::TCD1304Device (
    unsigned int period = CLK_DEFAULT) [inline]
```

Here is the call graph for this function:

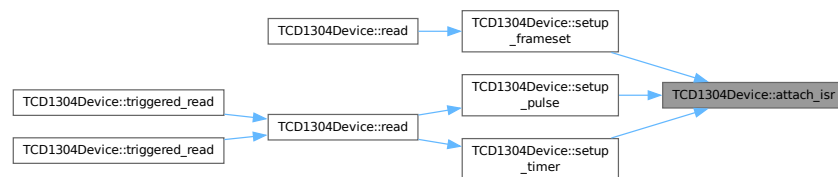


### 4.3.3 Member Function Documentation

#### 4.3.3.1 attach\_isr()

```
void TCD1304Device::attach_isr (
    SubModule * p,
    uint16_t cmpf_mask,
    void(* isr_f )()) [inline]
```

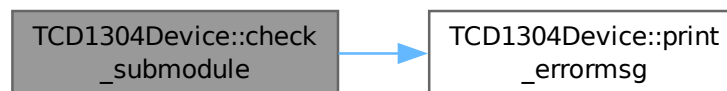
Here is the caller graph for this function:



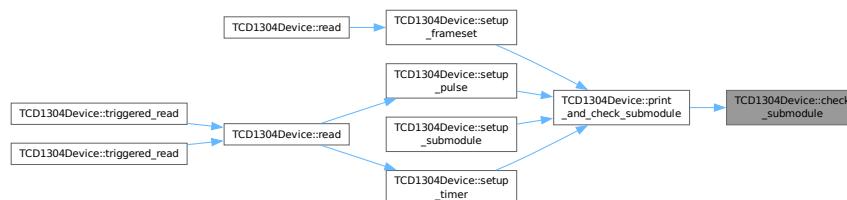
#### 4.3.3.2 check\_submodule()

```
bool TCD1304Device::check_submodule (
    SubModule * p) [inline]
```

Here is the call graph for this function:



Here is the caller graph for this function:



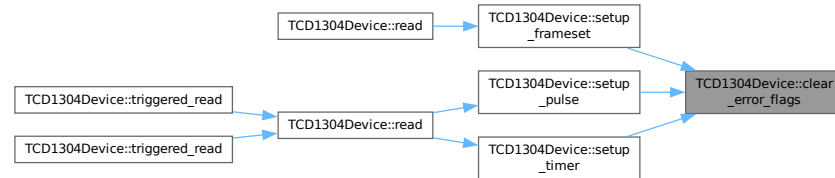
#### 4.3.3.3 clear\_buypin()

```
static void TCD1304Device::clear_buypin () [inline], [static]
```

#### 4.3.3.4 clear\_error\_flags()

```
static void TCD1304Device::clear_error_flags () [inline], [static]
```

Here is the caller graph for this function:



#### 4.3.3.5 clear\_frames\_completed\_callback()

```
void TCD1304Device::clear_frames_completed_callback () [inline]
```

#### 4.3.3.6 clear\_framesets\_completed\_callback()

```
void TCD1304Device::clear_framesets_completed_callback () [inline]
```

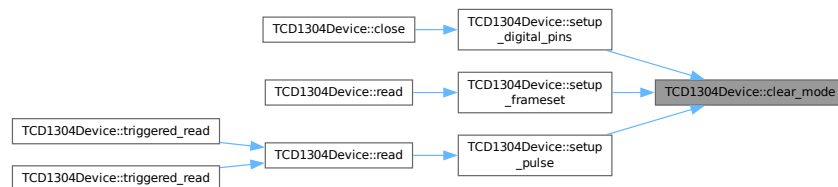
#### 4.3.3.7 clear\_ldok()

```
void TCD1304Device::clear_ldok (
    uint8_t mask,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.8 clear\_mode()

```
static void TCD1304Device::clear_mode () [inline], [static]
```

Here is the caller graph for this function:



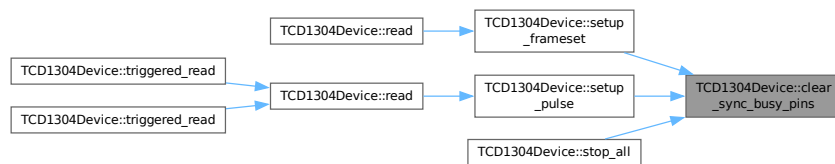
#### 4.3.3.9 clear\_run()

```
void TCD1304Device::clear_run (
    uint8_t mask,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.10 clear\_sync\_busy\_pins()

```
static void TCD1304Device::clear_sync_busy_pins () [inline], [static]
```

Here is the caller graph for this function:



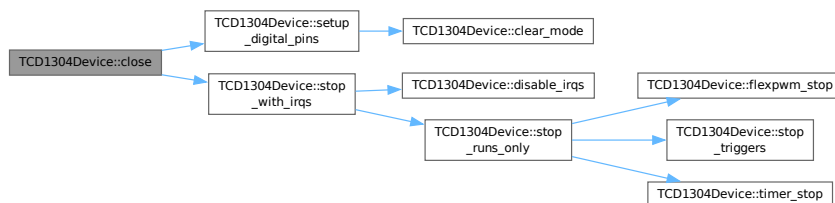
#### 4.3.3.11 clear\_synccpin()

```
static void TCD1304Device::clear_synccpin () [inline], [static]
```

#### 4.3.3.12 close()

```
static void TCD1304Device::close () [inline], [static]
```

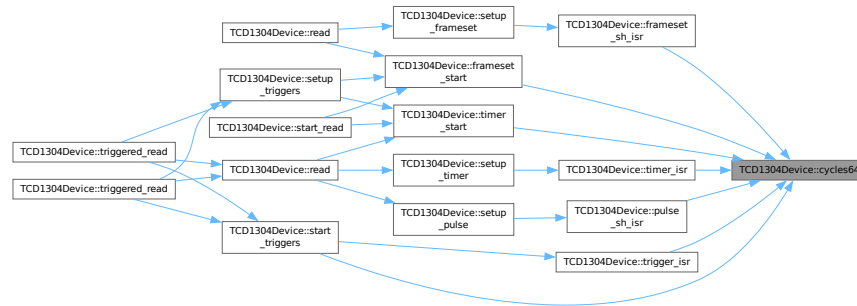
Here is the call graph for this function:



#### 4.3.3.13 cycles64()

```
static uint64_t TCD1304Device::cycles64 () [inline], [static]
```

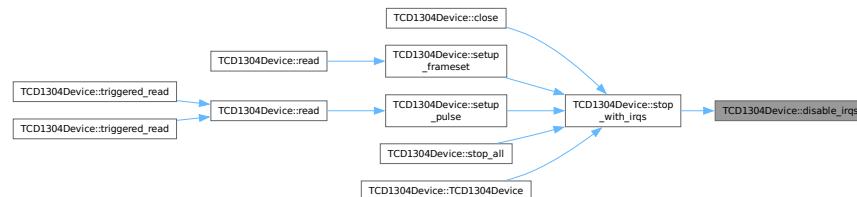
Here is the caller graph for this function:



#### 4.3.3.14 disable\_irqs()

```
static void TCD1304Device::disable_irqs () [inline], [static]
```

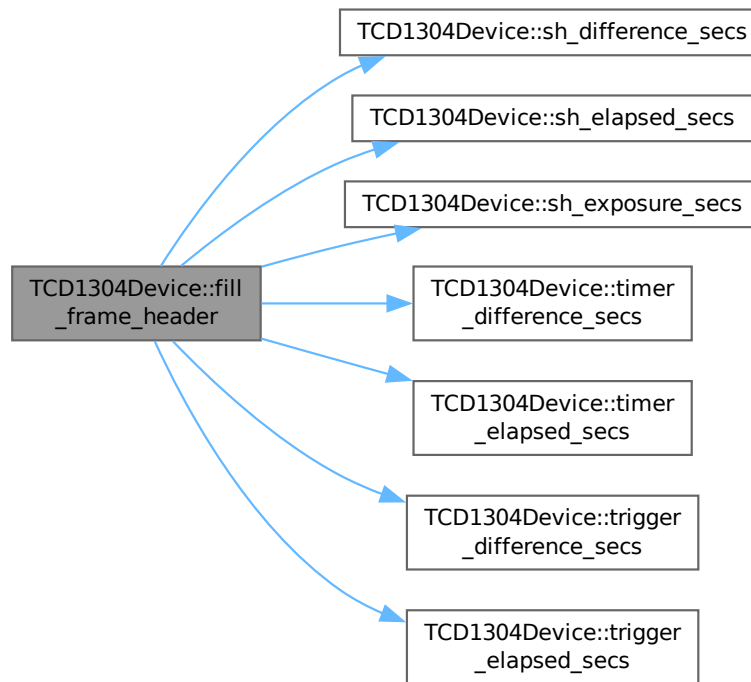
Here is the caller graph for this function:



#### 4.3.3.15 fill\_frame\_header()

```
static void TCD1304Device::fill_frame_header (
    Frame_Header * p) [inline], [static]
```

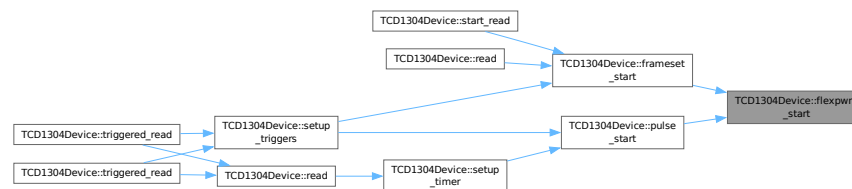
Here is the call graph for this function:



#### 4.3.3.16 flexpwm\_start()

```
static void TCD1304Device::flexpwm_start () [inline], [static]
```

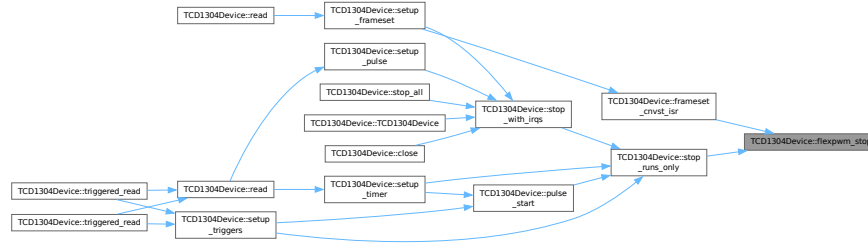
Here is the caller graph for this function:



#### 4.3.3.17 flexpwm\_stop()

```
static void TCD1304Device::flexpwm_stop () [inline], [static]
```

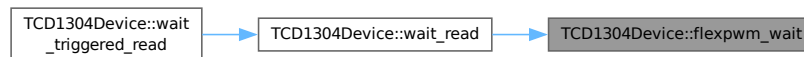
Here is the caller graph for this function:



#### 4.3.3.18 flexpwm\_wait()

```
bool TCD1304Device::flexpwm_wait (
    float timeout_ = 1.,
    float timestep_ = 0.01,
    bool verbose = false) [inline]
```

Here is the caller graph for this function:



#### 4.3.3.19 force()

```
void TCD1304Device::force (
    uint8_t submod,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.20 frameset\_arm()

```
static void TCD1304Device::frameset_arm () [inline], [static]
```

Here is the caller graph for this function:

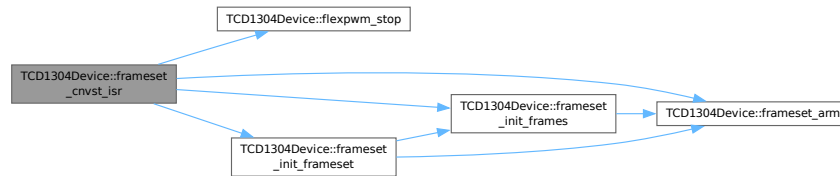




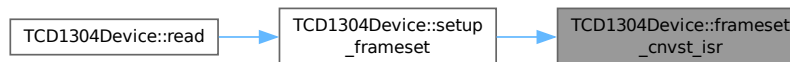
#### 4.3.3.21 frameset\_cnvst\_isr()

```
static void TCD1304Device::frameset_cnvst_isr () [inline], [static]
```

Here is the call graph for this function:



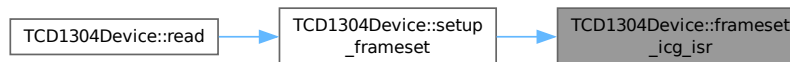
Here is the caller graph for this function:



#### 4.3.3.22 frameset\_icg\_isr()

```
static void TCD1304Device::frameset_icg_isr () [inline], [static]
```

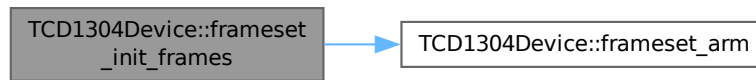
Here is the caller graph for this function:



#### 4.3.3.23 frameset\_init\_frames()

```
static void TCD1304Device::frameset_init_frames () [inline], [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.3.24 frameset\_init\_frameset()

```
static void TCD1304Device::frameset_init_frameset () [inline], [static]
```

Here is the call graph for this function:



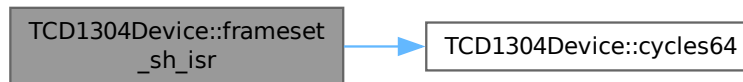
Here is the caller graph for this function:



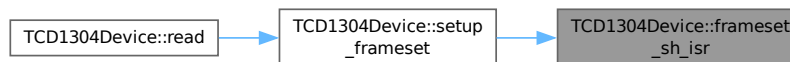
#### 4.3.3.25 frameset\_sh\_isr()

```
static void TCD1304Device::frameset_sh_isr () [inline], [static]
```

Here is the call graph for this function:



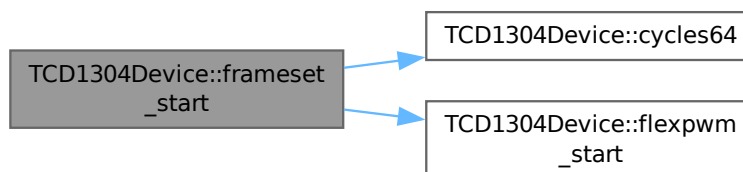
Here is the caller graph for this function:



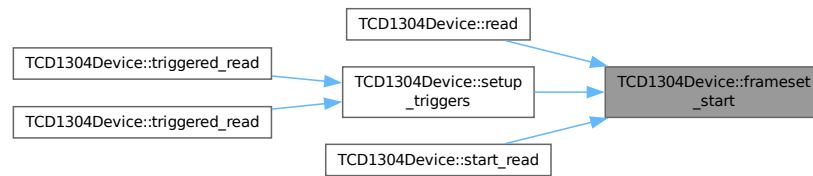
#### 4.3.3.26 frameset\_start()

```
static void TCD1304Device::frameset_start () [inline], [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.3.27 load\_frames\_completed\_callback()

```
void TCD1304Device::load_frames_completed_callback (
    void(* callback )(),
    unsigned int nframes = 0) [inline]
```

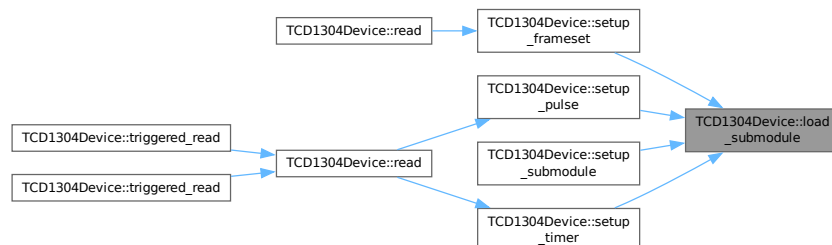
#### 4.3.3.28 load\_framesets\_completed\_callback()

```
void TCD1304Device::load_framesets_completed_callback (
    void(* callback )(),
    unsigned int nsets = 0) [inline]
```

#### 4.3.3.29 load\_submodule()

```
void TCD1304Device::load_submodule (
    SubModule * p) [inline]
```

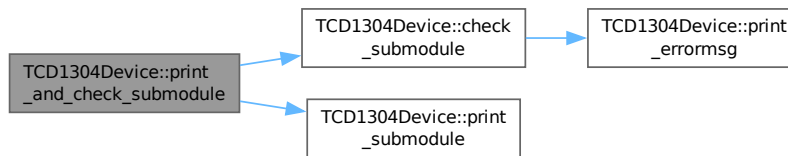
Here is the caller graph for this function:



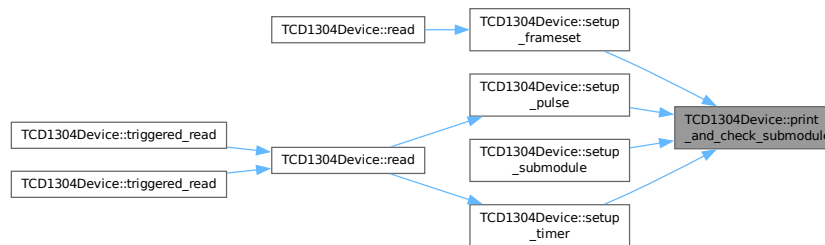
#### 4.3.3.30 print\_and\_check\_submodule()

```
bool TCD1304Device::print_and_check_submodule (
    SubModule * p) [inline]
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.3.31 print\_counters()

```
static void TCD1304Device::print_counters () [inline], [static]
```

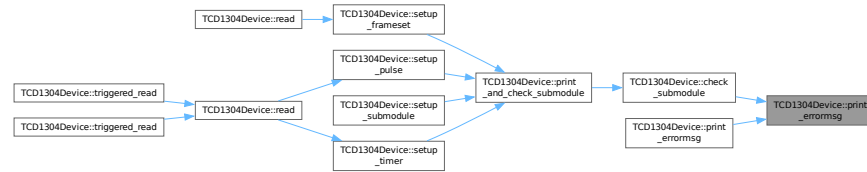
Here is the caller graph for this function:



#### 4.3.3.32 print\_errormsg() [1/2]

```
static void TCD1304Device::print_errormsg (
    const char * name,
    const char * errmsg) [inline], [static]
```

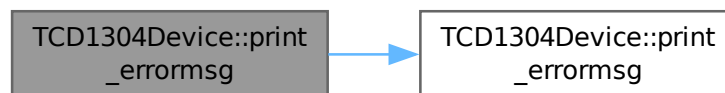
Here is the caller graph for this function:



#### 4.3.3.33 print\_errormsg() [2/2]

```
static void TCD1304Device::print_errormsg (
    SubModule * p,
    const char * errmsg) [inline], [static]
```

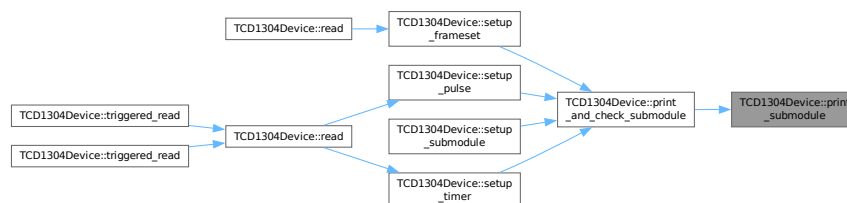
Here is the call graph for this function:



#### 4.3.3.34 print\_submodule()

```
void TCD1304Device::print_submodule (
    SubModule * p) [inline]
```

Here is the caller graph for this function:



#### 4.3.3.35 printbits16\_()

```
void TCD1304Device::printbits16_ (
    uint16_t u16,
    int bits) [inline]
```

Here is the caller graph for this function:



#### 4.3.3.36 pulse\_arm()

```
static void TCD1304Device::pulse_arm () [inline], [static]
```

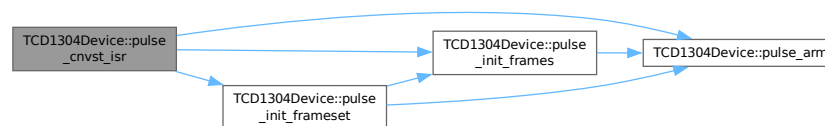
Here is the caller graph for this function:



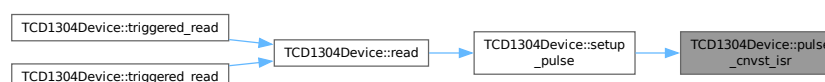
#### 4.3.3.37 pulse\_cnvt\_isr()

```
static void TCD1304Device::pulse_cnvt_isr () [inline], [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.3.38 pulse\_icg\_isr()

```
static void TCD1304Device::pulse_icg_isr () [inline], [static]
```

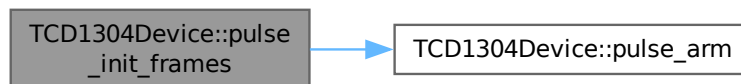
Here is the caller graph for this function:



#### 4.3.3.39 pulse\_init\_frames()

```
static void TCD1304Device::pulse_init_frames () [inline], [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:

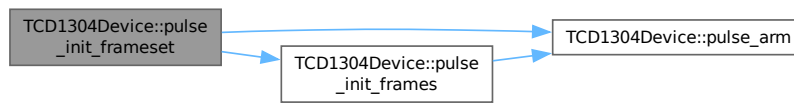


#### 4.3.3.40 pulse\_init\_frameset()

```
static void TCD1304Device::pulse_init_frameset () [inline], [static]
```



Here is the call graph for this function:



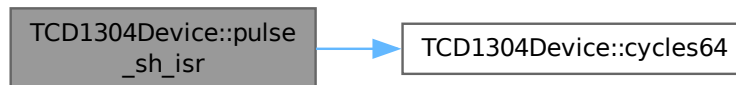
Here is the caller graph for this function:



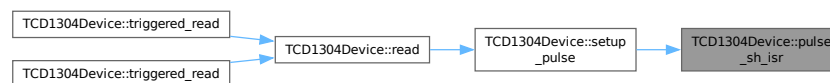
#### 4.3.3.41 pulse\_sh\_isr()

```
static void TCD1304Device::pulse_sh_isr () [inline], [static]
```

Here is the call graph for this function:



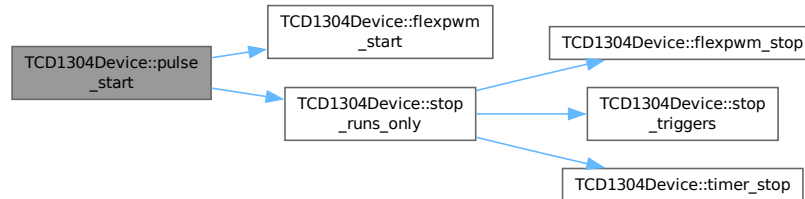
Here is the caller graph for this function:



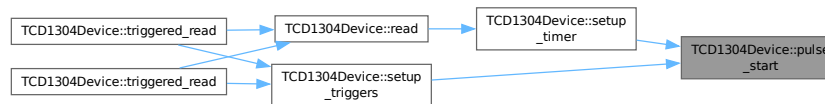
#### 4.3.3.42 pulse\_start()

```
static void TCD1304Device::pulse_start () [inline], [static]
```

Here is the call graph for this function:



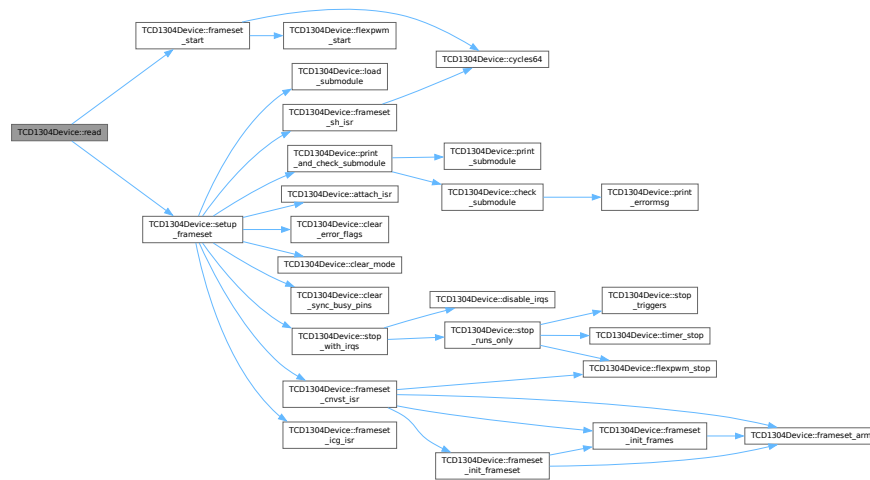
Here is the caller graph for this function:



#### 4.3.3.43 read() [1/2]

```
bool TCD1304Device::read (
    uint nframes,
    float exposure,
    float frame_interval,
    uint16_t * bufferp,
    void(* frame_callbackf )(),
    void(* frameset_callbackf )(),
    void(* completion_callbackf )(),
    void(* setup_callbackf )(),
    bool start = true) [inline]
```

Here is the call graph for this function:

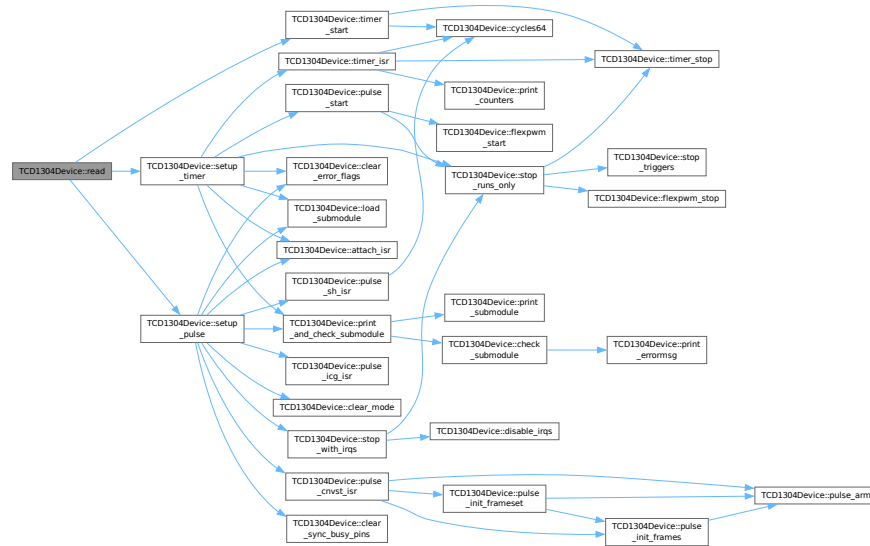


#### 4.3.3.44 read() [2/2]

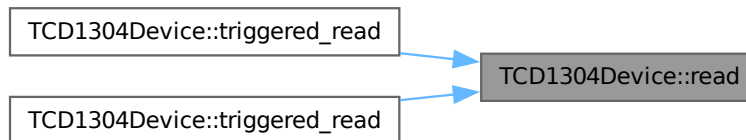
```

bool TCD1304Device::read (
    uint nframes,
    float exposure,
    uint16_t * bufferp,
    void(* frame_callbackf )(),
    void(* frameset_callbackf )(),
    void(* completion_callbackf )(),
    void(* setup_callbackf )(),
    bool start = true) [inline]
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.3.45 register\_dump()

```

void TCD1304Device::register_dump (
    IMXRT_FLEXPWM_t *const q = flexpwm,
    uint8_t mask = 0xFF) [inline]
  
```

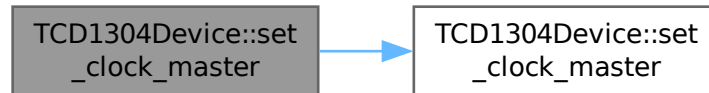
Here is the call graph for this function:



**4.3.3.46 set\_clock\_master()** [1/2]

```
void TCD1304Device::set_clock_master (  
    SubModule * p) [inline]
```

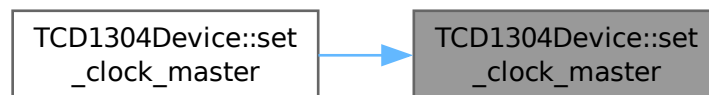
Here is the call graph for this function:

**4.3.3.47 set\_clock\_master()** [2/2]

```
void TCD1304Device::set_clock_master (  
    uint8_t submod,  
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

Low level functions for command line register access.

Here is the caller graph for this function:

**4.3.3.48 set\_clock\_slave()** [1/2]

```
void TCD1304Device::set_clock_slave (  
    SubModule * p) [inline]
```

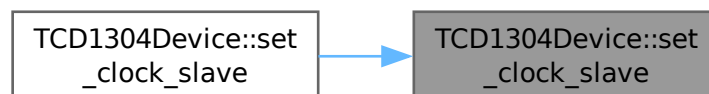
Here is the call graph for this function:



#### 4.3.3.49 set\_clock\_slave() [2/2]

```
void TCD1304Device::set_clock_slave (  
    uint8_t submod,  
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

Here is the caller graph for this function:



#### 4.3.3.50 set\_clock\_sync() [1/2]

```
void TCD1304Device::set_clock_sync (  
    SubModule * p) [inline]
```

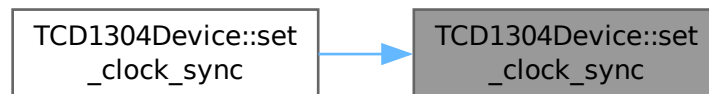
Here is the call graph for this function:



#### 4.3.3.51 set\_clock\_sync() [2/2]

```
void TCD1304Device::set_clock_sync (
    uint8_t submod,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

Here is the caller graph for this function:



#### 4.3.3.52 set\_init()

```
void TCD1304Device::set_init (
    uint8_t submod,
    uint16_t value,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.53 set\_ldok()

```
void TCD1304Device::set_ldok (
    uint8_t mask,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.54 set\_outen()

```
uint16_t TCD1304Device::set_outen (
    uint16_t mask16,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.55 set\_outen\_off()

```
uint16_t TCD1304Device::set_outen_off (
    uint16_t mask16,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.56 set\_outen\_on()

```
uint16_t TCD1304Device::set_outen_on (
    uint16_t mask16,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.57 set\_outenA\_on()

```
uint16_t TCD1304Device::set_outenA_on (
    uint8_t mask8,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.58 set\_prescale()

```
void TCD1304Device::set_prescale (
    uint8_t submod,
    uint8_t prescale = 0,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.59 set\_run()

```
void TCD1304Device::set_run (
    uint8_t mask,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.60 set\_val0()

```
void TCD1304Device::set_val0 (
    uint8_t submod,
    uint16_t value,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.61 set\_val1()

```
void TCD1304Device::set_val1 (
    uint8_t submod,
    uint16_t value,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.62 set\_val2()

```
void TCD1304Device::set_val2 (
    uint8_t submod,
    uint16_t value,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```



#### 4.3.3.63 set\_val3()

```
void TCD1304Device::set_val3 (
    uint8_t submod,
    uint16_t value,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

#### 4.3.3.64 set\_val4()

```
void TCD1304Device::set_val4 (
    uint8_t submod,
    uint16_t value,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

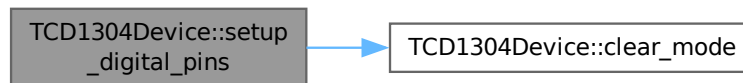
#### 4.3.3.65 set\_val5()

```
void TCD1304Device::set_val5 (
    uint8_t submod,
    uint16_t value,
    IMXRT_FLEXPWM_t * q = flexpwm) [inline]
```

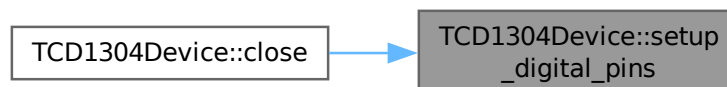
#### 4.3.3.66 setup\_digital\_pins()

```
static void TCD1304Device::setup_digital_pins () [inline], [static]
```

Here is the call graph for this function:



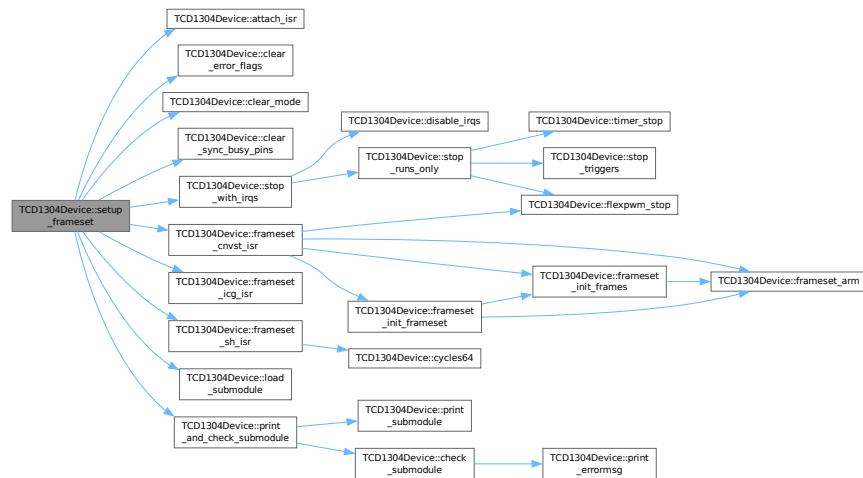
Here is the caller graph for this function:



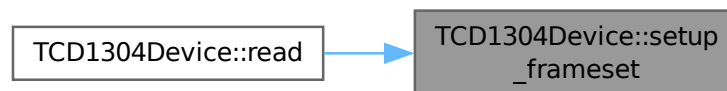
#### 4.3.3.67 setup\_frameset()

```
bool TCD1304Device::setup_frameset (
    float clk_secs,
    float sh_secs,
    float sh_offset_secs,
    float icg_secs,
    float icg_offset_secs,
    float exposure_secs,
    float frame_interval_secs,
    unsigned int nframes,
    uint16_t * buffer,
    unsigned int nbuffer,
    void(* callbackf )()) [inline]
```

Here is the call graph for this function:



Here is the caller graph for this function:



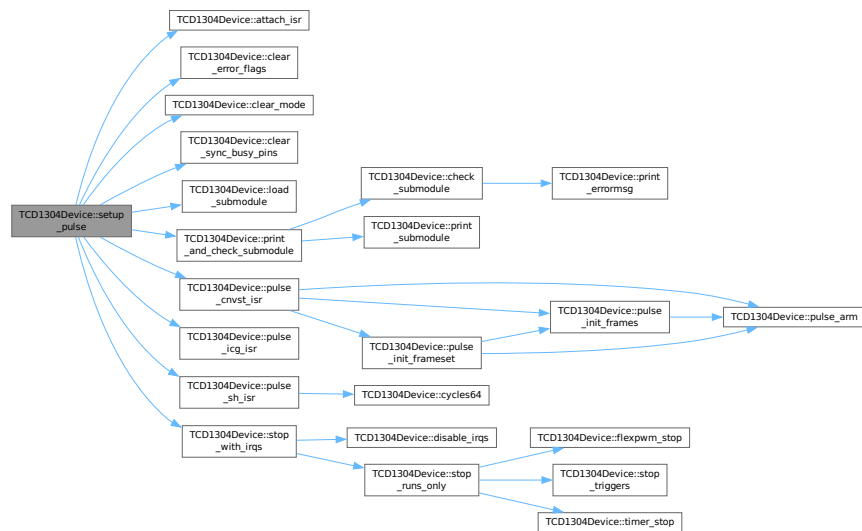
## 4.3.3.68 setup\_pulse()

```

bool TCD1304Device::setup_pulse (
    float clk_secs,
    float sh_secs,
    float sh_offset_secs,
    float icg_secs,
    float icg_offset_secs,
    uint16_t * buffer,
    unsigned int nbuffer,
    void(* callbackf )()) [inline]

```

Here is the call graph for this function:



Here is the caller graph for this function:



## 4.3.3.69 setup\_submodule()

```

bool TCD1304Device::setup_submodule (
    SubModule * p,

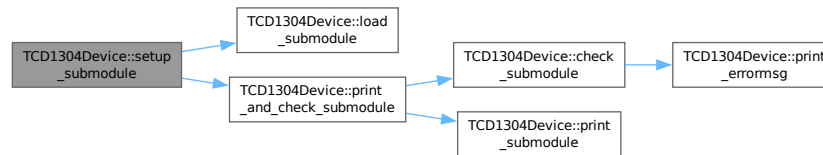
```

```

uint8_t prescale,
uint16_t period_counts,
uint16_t onA_counts,
uint16_t offA_counts,
bool invertA,
uint16_t onB_counts,
uint16_t offB_counts,
bool invertB,
uint16_t ctrl2_mask) [inline]

```

Here is the call graph for this function:



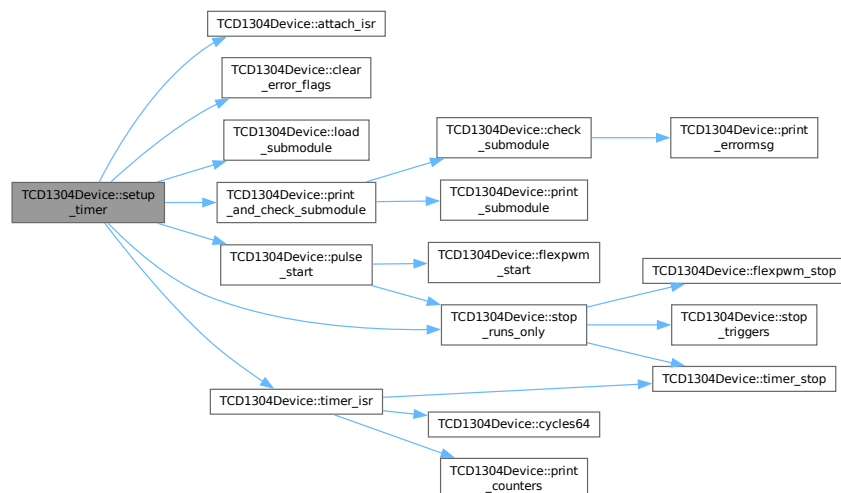
#### 4.3.3.70 setup\_timer()

```

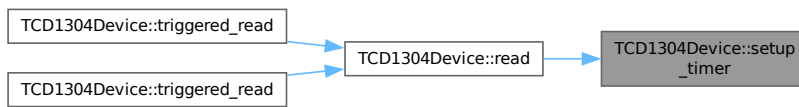
bool TCD1304Device::setup_timer (
    float exposure_secs,
    float exposure_offset_secs,
    unsigned int ncounts = 0) [inline]

```

Here is the call graph for this function:



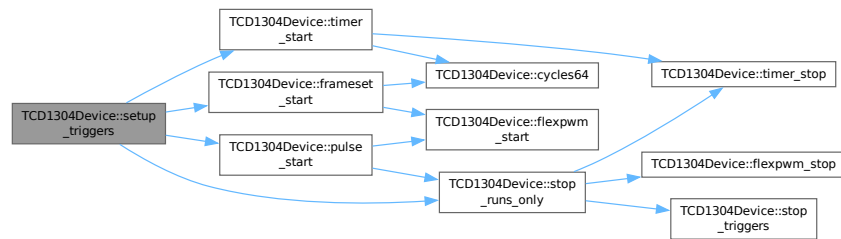
Here is the caller graph for this function:



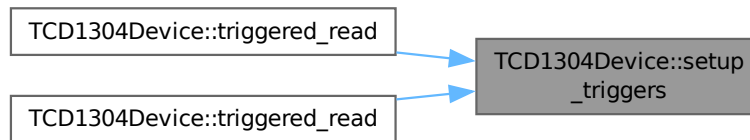
#### 4.3.3.71 setup\_triggers()

```
bool TCD1304Device::setup_triggers (
    unsigned int ncounts = 0) [inline]
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.3.3.72 sh\_difference\_secs()

```
static double TCD1304Device::sh_difference_secs () [inline], [static]
```

Here is the caller graph for this function:



#### 4.3.3.73 sh\_elapsed\_secs()

```
static double TCD1304Device::sh_elapsed_secs () [inline], [static]
```

Here is the caller graph for this function:



#### 4.3.3.74 sh\_exposure\_secs()

```
static double TCD1304Device::sh_exposure_secs () [inline], [static]
```

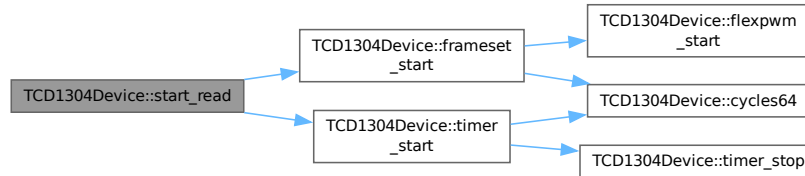
Here is the caller graph for this function:



#### 4.3.3.75 start\_read()

```
bool TCD1304Device::start_read () [inline]
```

Here is the call graph for this function:



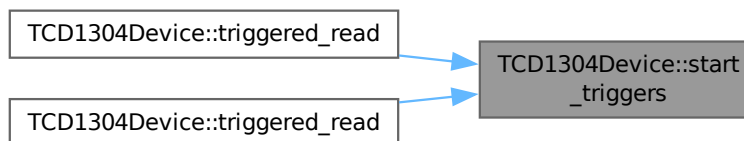
#### 4.3.3.76 start\_triggers()

```
bool TCD1304Device::start_triggers () [inline]
```

Here is the call graph for this function:



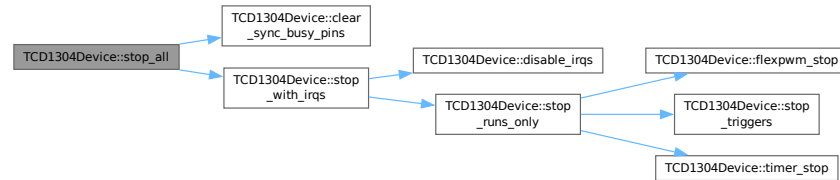
Here is the caller graph for this function:



#### 4.3.3.77 stop\_all()

```
void TCD1304Device::stop_all () [inline]
```

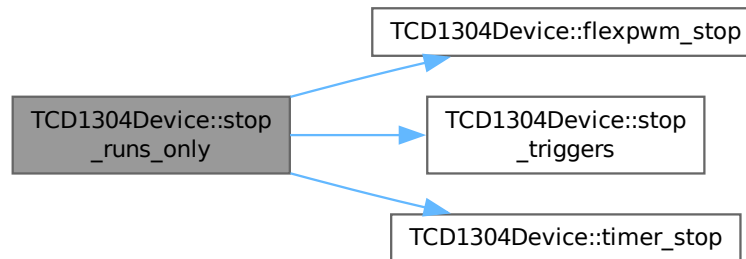
Here is the call graph for this function:



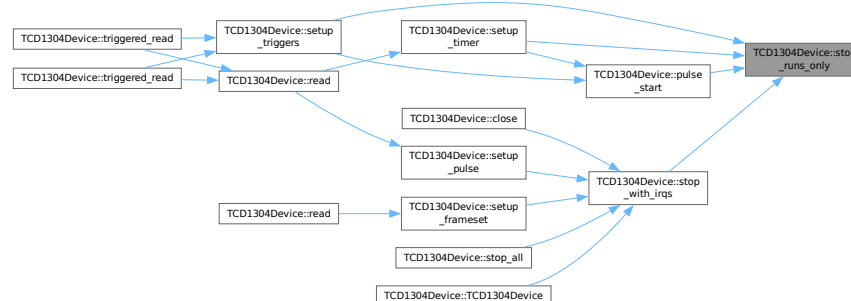
#### 4.3.3.78 stop\_runs\_only()

```
static void TCD1304Device::stop_runs_only () [inline], [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:





```
static void TCD1304Device::stop_triggers () [inline], [static]
```

```

graph LR
    A[TCD1304Device::triggered_read] --> B[TCD1304Device::setup_triggers]
    A --> C[TCD1304Device::read]
    B --> D[TCD1304Device::timer]
    B --> E[TCD1304Device::pulse_start]
    C --> D
    C --> F[TCD1304Device::close]
    C --> G[TCD1304Device::setup_pulse]
    C --> H[TCD1304Device::setup_frameset]
    D --> I[TCD1304Device::stop_runs_only]
    E --> I
    F --> J[TCD1304Device::stop_with_irqs]
    G --> J
    H --> J
    I --> K[TCD1304Device::stop_triggers]
    L[TCD1304Device::read] --> H
    M[TCD1304Device::stop_all] --> J
    N[TCD1304Device::TCD1304Device] --> J

```

```
static void TCD1304Device::stop_with_irqs () [inline], [static]
```

```

graph LR
    A[TCD1304Device::stop_with_irqs] --> B[TCD1304Device::disable_irqs]
    A --> C[TCD1304Device::stop_runs_only]
    B --> D[TCD1304Device::flexpwm_stop]
    C --> D
    C --> E[TCD1304Device::stop_triggers]
    C --> F[TCD1304Device::timer_stop]
  
```

The control flow graph for `TCD1304Device::stop_with_irqs` shows the following structure:

- Start Node:** `TCD1304Device::stop_with_irqs` (shaded box).
- Branch 1:** An arrow points to `TCD1304Device::disable_irqs`.
- Branch 2:** An arrow points to `TCD1304Device::stop_runs_only`.
- From `disable_irqs`:** An arrow points to `TCD1304Device::flexpwm_stop`.
- From `stop_runs_only`:**
  - An arrow points to `TCD1304Device::flexpwm_stop`.
  - An arrow points to `TCD1304Device::stop_triggers`.
  - An arrow points to `TCD1304Device::timer_stop`.

```

stateDiagram-v2
    [*] --> TCD1304Device::triggered_read
    TCD1304Device::triggered_read --> TCD1304Device::read : read
    TCD1304Device::read --> TCD1304Device::read : read
    TCD1304Device::read --> TCD1304Device::close : close
    TCD1304Device::read --> TCD1304Device::setup_frameset : read
    TCD1304Device::read --> TCD1304Device::setup_pulse : read
    TCD1304Device::read --> TCD1304Device::stop_all : read
    TCD1304Device::close --> TCD1304Device::stop_with_irqs : stop_with_irqs
    TCD1304Device::setup_frameset --> TCD1304Device::stop_with_irqs : stop_with_irqs
    TCD1304Device::setup_pulse --> TCD1304Device::stop_with_irqs : stop_with_irqs
    TCD1304Device::stop_all --> TCD1304Device::stop_with_irqs : stop_with_irqs
    TCD1304Device::TCD1304Device --> TCD1304Device::stop_with_irqs : stop_with_irqs
    TCD1304Device::stop_with_irqs --> [*]
  
```

#### 4.3.3.81 timer\_difference\_secs()

```
static double TCD1304Device::timer_difference_secs () [inline], [static]
```

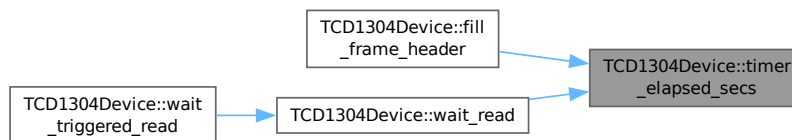
Here is the caller graph for this function:



#### 4.3.3.82 timer\_elapsed\_secs()

```
static double TCD1304Device::timer_elapsed_secs () [inline], [static]
```

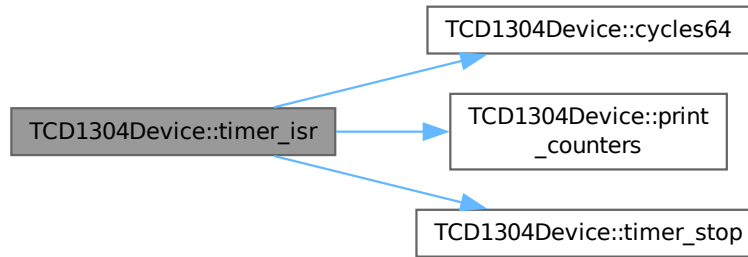
Here is the caller graph for this function:



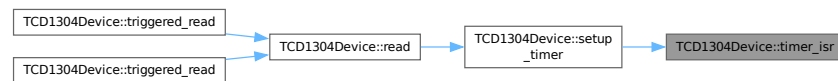
#### 4.3.3.83 timer\_isr()

```
static void TCD1304Device::timer_isr () [inline], [static]
```

Here is the call graph for this function:



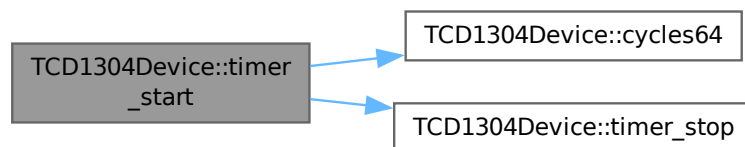
Here is the caller graph for this function:



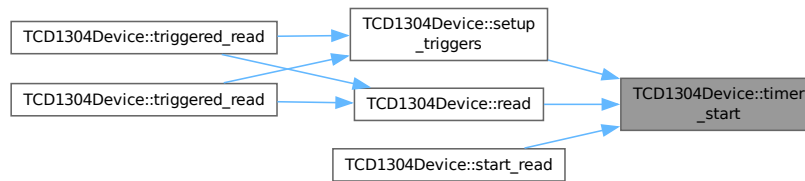
#### 4.3.3.84 timer\_start()

```
static void TCD1304Device::timer_start () [inline], [static]
```

Here is the call graph for this function:



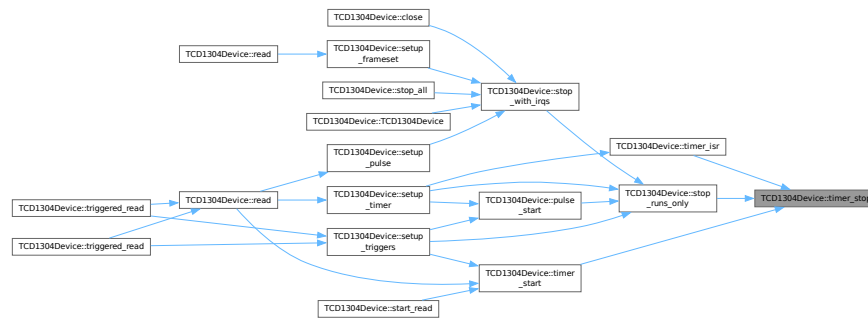
Here is the caller graph for this function:



#### 4.3.3.85 timer\_stop()

```
static void TCD1304Device::timer_stop () [inline], [static]
```

Here is the caller graph for this function:



#### 4.3.3.86 timer\_stop\_with\_irq()

```
static void TCD1304Device::timer_stop_with_irq () [inline], [static]
```

#### 4.3.3.87 timer\_wait()

```
bool TCD1304Device::timer_wait (
    float timeout_ = 1.,
    float timestep_ = 0.01,
    bool verbose = false) [inline]
```

Here is the caller graph for this function:



#### 4.3.3.88 toggle\_busypin()

```
static void TCD1304Device::toggle_busypin () [inline], [static]
```

#### 4.3.3.89 toggle\_syncpin()

```
static void TCD1304Device::toggle_syncpin () [inline], [static]
```

#### 4.3.3.90 trigger\_difference\_secs()

```
static double TCD1304Device::trigger_difference_secs () [inline], [static]
```

Here is the caller graph for this function:



#### 4.3.3.91 trigger\_elapsed\_secs()

```
static double TCD1304Device::trigger_elapsed_secs () [inline], [static]
```

Here is the caller graph for this function:



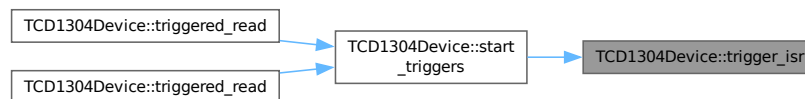
#### 4.3.3.92 trigger\_isr()

```
static void TCD1304Device::trigger_isr () [inline], [static]
```

Here is the call graph for this function:



Here is the caller graph for this function:

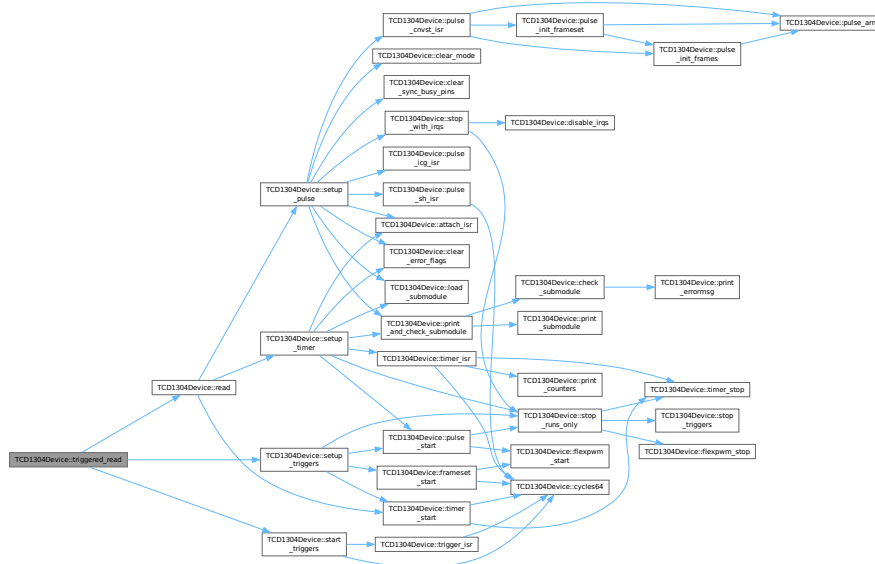


#### 4.3.3.93 triggered\_read() [1/2]

```
bool TCD1304Device::triggered_read (
    uint ntriggers,
    uint nframes,
    float exposure,
    float interval,
    uint16_t * bufferp,
    void(* frame_callbackf )(),
    void(* frameset_callbackf )(),
    void(* completion_callbackf )(),
    void(* setup_callbackf )(),
    bool start = true) [inline]
```

[illegible]

```
bool TCD1304Device::triggered_read (
    uint ntriggers,
    uint nframes,
    float exposure,
    uint16_t * bufferp,
    void(* frame_callbackf )(),
    void(* frameset_callbackf )(),
    void(* completion_callbackf )(),
    void(* setup_callbackf )(),
    bool start = true) [inline]
```

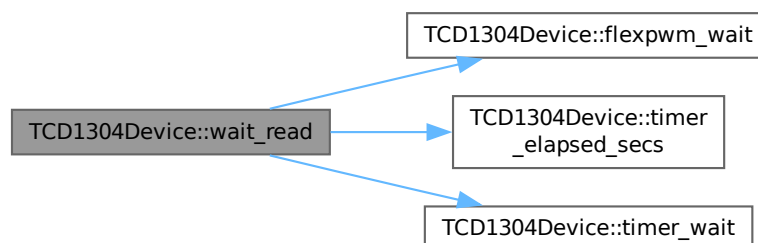


```
static void TCD1304Device::update_read_buffer (
    uint16_t * buffer) [inline], [static]
```

#### 4.3.3.96 wait\_read()

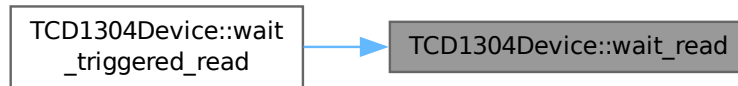
```
bool TCD1304Device::wait_read (
    float timeout = 0.,
    float timestep = 0.01,
    bool verbose = false) [inline]
```

Here is the call graph for this function:





Here is the caller graph for this function:

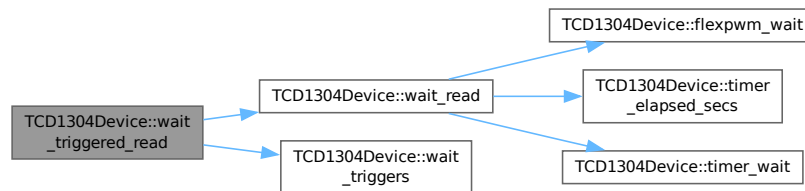


#### 4.3.3.97 wait\_triggered\_read()

```

bool TCD1304Device::wait_triggered_read (
    float timeout = 1.,
    float timestep = 0.01,
    bool verbose = false) [inline]
  
```

Here is the call graph for this function:

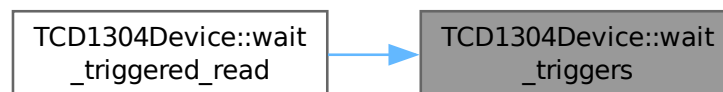


#### 4.3.3.98 wait\_triggers()

```

bool TCD1304Device::wait_triggers (
    float timeout_ = 1.,
    float timestep_ = 0.01,
    bool verbose = false) [inline]
  
```

Here is the caller graph for this function:



### 4.3.4 Member Data Documentation

#### 4.3.4.1 busytoggled

```
bool TCD1304Device::busytoggled = false [inline], [static]
```

#### 4.3.4.2 clk

```
SubModule TCD1304Device::clk = {"clk", CLK_SUBMODULE, CLK_MASK, CLK_PIN, CLK_MUXVAL, 0xFF, 0, CLK_IRQ,  
&IMXRT_FLEXPWM2} [inline], [static]
```

#### 4.3.4.3 cnvst

```
SubModule TCD1304Device::cnvst = {"cnvst", CNVST_SUBMODULE, CNVST_MASK, 0xFF, 0, 0xFF, 0, CNVST_IRQ,  
&IMXRT_FLEXPWM2} [inline], [static]
```

#### 4.3.4.4 cnvst\_counter

```
unsigned int TCD1304Device::cnvst_counter = 0 [inline], [static]
```

#### 4.3.4.5 cnvst\_extra\_delay\_counts

```
uint16_t TCD1304Device::cnvst_extra_delay_counts = 1 [inline], [static]
```

#### 4.3.4.6 error\_flag

```
bool TCD1304Device::error_flag = false [inline], [static]
```

#### 4.3.4.7 flexpwm

```
IMXRT_FLEXPWM_t* const TCD1304Device::flexpwm = &IMXRT_FLEXPWM2 [inline], [static]
```

#### 4.3.4.8 flexpwm\_running

```
bool TCD1304Device::flexpwm_running = false [inline], [static]
```

#### 4.3.4.9 frame\_counter

```
unsigned int TCD1304Device::frame_counter = 0 [inline], [static]
```

#### 4.3.4.10 frame\_counts

```
unsigned int TCD1304Device::frame_counts = 0 [inline], [static]
```

#### 4.3.4.11 frames\_completed\_callback

```
void(* TCD1304Device::frames_completed_callback) ()=0 [inline], [static]
```

#### 4.3.4.12 frameset\_armed

```
bool TCD1304Device::frameset_armed = false [inline], [static]
```

#### 4.3.4.13 frameset\_counter

```
unsigned int TCD1304Device::frameset_counter = 0 [inline], [static]
```

#### 4.3.4.14 frameset\_counts

```
unsigned int TCD1304Device::frameset_counts = 0 [inline], [static]
```

#### 4.3.4.15 framesets\_completed\_callback

```
void(* TCD1304Device::framesets_completed_callback) ()=0 [inline], [static]
```

#### 4.3.4.16 icg

```
SubModule TCD1304Device::icg = {"icg", ICG_SUBMODULE, ICG_MASK, ICG_PIN, ICG_MUXVAL, 0xFF, 0, ICG_IRQ,  
&IMXRT_FLEXPWM2} [inline], [static]
```

#### 4.3.4.17 icg\_counter

```
unsigned int TCD1304Device::icg_counter = 0 [inline], [static]
```

#### 4.3.4.18 mode

```
TCD1304_Mode_t TCD1304Device::mode = NOTCONFIGURED [inline], [static]
```

#### 4.3.4.19 `oops_flag`

```
bool TCD1304Device::oops_flag = false [inline], [static]
```

#### 4.3.4.20 `pulse_armed`

```
bool TCD1304Device::pulse_armed = false [inline], [static]
```

Configure the flexpwm for single pulse, use this with a timer.

#### 4.3.4.21 `read_buffer`

```
uint16_t* TCD1304Device::read_buffer = 0 [inline], [static]
```

#### 4.3.4.22 `read_callback`

```
void(* TCD1304Device::read_callback) ()=0 [inline], [static]
```

#### 4.3.4.23 `read_counter`

```
unsigned int TCD1304Device::read_counter = 0 [inline], [static]
```

#### 4.3.4.24 `read_counts`

```
unsigned int TCD1304Device::read_counts = 0 [inline], [static]
```

#### 4.3.4.25 `read_expected_time`

```
float TCD1304Device::read_expected_time = 0. [inline], [static]
```

#### 4.3.4.26 `read_pointer`

```
uint16_t* TCD1304Device::read_pointer = 0 [inline], [static]
```

#### 4.3.4.27 `sh`

```
SubModule TCD1304Device::sh = {"sh", SH_SUBMODULE, SH_MASK, SH_PIN, SH_MUXVAL, 0xFF, 0, SH_IRQ,  
&IMXRT_FLEXPWM2} [inline], [static]
```

#### 4.3.4.28 sh\_clearing\_counter

```
unsigned int TCD1304Device::sh_clearing_counter = 0 [inline], [static]
```

#### 4.3.4.29 sh\_clearing\_counts

```
unsigned int TCD1304Device::sh_clearing_counts = SH_CLEARING_DEFAULT [inline], [static]
```

#### 4.3.4.30 sh\_counter

```
unsigned int TCD1304Device::sh_counter = 0 [inline], [static]
```

#### 4.3.4.31 sh\_counts\_per\_icg

```
unsigned int TCD1304Device::sh_counts_per_icg = 0 [inline], [static]
```

#### 4.3.4.32 sh\_cyccnt64\_exposure

```
uint64_t TCD1304Device::sh_cyccnt64_exposure = 0 [inline], [static]
```

#### 4.3.4.33 sh\_cyccnt64\_now

```
uint64_t TCD1304Device::sh_cyccnt64_now = 0 [inline], [static]
```

#### 4.3.4.34 sh\_cyccnt64\_prev

```
uint64_t TCD1304Device::sh_cyccnt64_prev = 0 [inline], [static]
```

#### 4.3.4.35 sh\_cyccnt64\_start

```
uint64_t TCD1304Device::sh_cyccnt64_start = 0 [inline], [static]
```

#### 4.3.4.36 sh\_short\_period\_counts

```
unsigned int TCD1304Device::sh_short_period_counts = 0 [inline], [static]
```

#### 4.3.4.37 skip\_one

```
bool TCD1304Device::skip_one = false [inline], [static]
```

#### 4.3.4.38 skip\_one\_reload

```
bool TCD1304Device::skip_one_reload = false [inline], [static]
```

#### 4.3.4.39 sync\_enabled

```
bool TCD1304Device::sync_enabled = true [inline], [static]
```

#### 4.3.4.40 sync\_pin

```
uint8_t TCD1304Device::sync_pin = SYNC_PIN [inline], [static]
```

#### 4.3.4.41 synctoggled

```
bool TCD1304Device::synctoggled = false [inline], [static]
```

#### 4.3.4.42 timer

```
SubModule TCD1304Device::timer = {"timer", TIMER_SUBMODULE, TIMER_MASK, 0xFF, 0, TIMER_PIN, TIMER_MUXVAL, TIMER_IRQ, &IMXRT_FLEXPWM4} [inline], [static]
```

#### 4.3.4.43 timer\_callback

```
void(* TCD1304Device::timer_callback) ()=0 [inline], [static]
```

#### 4.3.4.44 timer\_cyccnt64\_now

```
uint64_t TCD1304Device::timer_cyccnt64_now = 0 [inline], [static]
```

#### 4.3.4.45 timer\_cyccnt64\_prev

```
uint64_t TCD1304Device::timer_cyccnt64_prev = 0 [inline], [static]
```

#### 4.3.4.46 timer\_cyccnt64\_start

```
uint64_t TCD1304Device::timer_cyccnt64_start = 0 [inline], [static]
```

**4.3.4.47 timer\_first\_time\_flag**

```
bool TCD1304Device::timer_first_time_flag = true [inline], [static]
```

**4.3.4.48 timer\_inner\_counter**

```
unsigned int TCD1304Device::timer_inner_counter = 0 [inline], [static]
```

**4.3.4.49 timer\_inner\_counts**

```
unsigned int TCD1304Device::timer_inner_counts = 0 [inline], [static]
```

**4.3.4.50 timer\_interframe\_min\_secs**

```
float TCD1304Device::timer_interframe_min_secs = 0. [inline], [static]
```

**4.3.4.51 timer\_interval\_secs**

```
float TCD1304Device::timer_interval_secs = .0 [inline], [static]
```

**4.3.4.52 timer\_outer\_counter**

```
unsigned int TCD1304Device::timer_outer_counter = 0 [inline], [static]
```

**4.3.4.53 timer\_outer\_counts**

```
unsigned int TCD1304Device::timer_outer_counts = 0 [inline], [static]
```

**4.3.4.54 timer\_period\_secs**

```
float TCD1304Device::timer_period_secs = 0 [inline], [static]
```

**4.3.4.55 timer\_running**

```
bool TCD1304Device::timer_running = false [inline], [static]
```

**4.3.4.56 timerflexpwm**

```
IMXRT_FLEXPWM_t* const TCD1304Device::timerflexpwm = &IMXRT_FLEXPWM4 [inline], [static]
```

#### 4.3.4.57 trigger\_attached

```
bool TCD1304Device::trigger_attached = false [inline], [static]
```

#### 4.3.4.58 trigger\_busy

```
bool TCD1304Device::trigger_busy = false [inline], [static]
```

#### 4.3.4.59 trigger\_callback

```
void(* TCD1304Device::trigger_callback) ()=0 [inline], [static]
```

#### 4.3.4.60 trigger\_counter

```
unsigned int TCD1304Device::trigger_counter = 0 [inline], [static]
```

#### 4.3.4.61 trigger\_counts

```
unsigned int TCD1304Device::trigger_counts = 1 [inline], [static]
```

#### 4.3.4.62 trigger\_cyccnt64\_now

```
uint64_t TCD1304Device::trigger_cyccnt64_now = 0 [inline], [static]
```

#### 4.3.4.63 trigger\_cyccnt64\_prev

```
uint64_t TCD1304Device::trigger_cyccnt64_prev = 0 [inline], [static]
```

#### 4.3.4.64 trigger\_cyccnt64\_start

```
uint64_t TCD1304Device::trigger_cyccnt64_start = 0 [inline], [static]
```

#### 4.3.4.65 trigger\_edge\_mode

```
uint8_t TCD1304Device::trigger_edge_mode = RISING [inline], [static]
```

#### 4.3.4.66 trigger\_mode

```
bool TCD1304Device::trigger_mode = false [inline], [static]
```



#### 4.3.4.67 trigger\_pin

```
uint8_t TCD1304Device::trigger_pin = TRIGGER_PIN [inline], [static]
```

#### 4.3.4.68 trigger\_pin\_mode

```
uint8_t TCD1304Device::trigger_pin_mode = INPUT [inline], [static]
```

The documentation for this class was generated from the following file:

- [TCD1304Device2.h](#)

## 4.4 usb\_string\_descriptor\_struct\_manufacturer Struct Reference

### Public Attributes

- uint8\_t [bLength](#)
- uint8\_t [bDescriptorType](#)
- uint16\_t [wString](#) [[thisMANUFACTURER\\_NAME\\_LEN](#)]

### 4.4.1 Member Data Documentation

#### 4.4.1.1 bDescriptorType

```
uint8_t usb_string_descriptor_struct_manufacturer::bDescriptorType
```

#### 4.4.1.2 bLength

```
uint8_t usb_string_descriptor_struct_manufacturer::bLength
```

#### 4.4.1.3 wString

```
uint16_t usb_string_descriptor_struct_manufacturer::wString [thisMANUFACTURER\_NAME\_LEN]
```

The documentation for this struct was generated from the following file:

- [TCD1304Device\\_Controller\\_251208.ino](#)

## 4.5 usb\_string\_descriptor\_struct\_product Struct Reference

### Public Attributes

- `uint8_t` [bLength](#)
- `uint8_t` [bDescriptorType](#)
- `uint16_t` [wString](#) [[thisPRODUCT\\_NAME\\_LEN](#)]

### 4.5.1 Member Data Documentation

#### 4.5.1.1 bDescriptorType

```
uint8_t usb_string_descriptor_struct_product::bDescriptorType
```

#### 4.5.1.2 bLength

```
uint8_t usb_string_descriptor_struct_product::bLength
```

#### 4.5.1.3 wString

```
uint16_t usb_string_descriptor_struct_product::wString[thisPRODUCT\_NAME\_LEN]
```

The documentation for this struct was generated from the following file:

- [TCD1304Device\\_Controller\\_251208.ino](#)

## 4.6 usb\_string\_descriptor\_struct\_serial\_number Struct Reference

### Public Attributes

- `uint8_t` [bLength](#)
- `uint8_t` [bDescriptorType](#)
- `uint16_t` [wString](#) [[thisPRODUCT\\_SERIAL\\_NUMBER\\_LEN](#)]

### 4.6.1 Member Data Documentation

#### 4.6.1.1 bDescriptorType

```
uint8_t usb_string_descriptor_struct_serial_number::bDescriptorType
```

#### 4.6.1.2 bLength

```
uint8_t usb_string_descriptor_struct_serial_number::bLength
```

#### 4.6.1.3 wString

```
uint16_t usb_string_descriptor_struct_serial_number::wString[thisPRODUCT\_SERIAL\_NUMBER\_LEN]
```

The documentation for this struct was generated from the following file:

- [TCD1304Device\\_Controller\\_251208.ino](#)

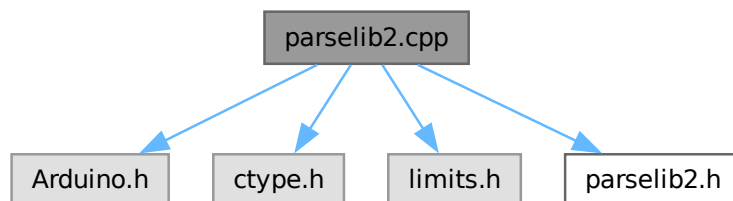
## Chapter 5

# File Documentation

### 5.1 parselib2.cpp File Reference

```
#include "Arduino.h"
#include <ctype.h>
#include <limits.h>
#include "parselib2.h"
```

Include dependency graph for parselib2.cpp:



#### Functions

- int [serialPrintf](#) (const char \*format,...)
- int [serialPrintlnf](#) (const char \*format,...)
- char \* [eow](#) (char \*s)
- char \* [eos](#) (char \*s)
- unsigned int [wordLength](#) (char \*s)
- char \* [nextWord](#) (char \*s)
- char \* [basenamef](#) (const char \*cs)
- unsigned int [countWords](#) (char \*s)
- bool [strMatch](#) (const char \*cs, const char \*key, char \*\*next)

- bool [strBool](#) (const char \*cs, bool \*b, char \*\*next)
- bool [strUInt8lim](#) (const char \*cs, uint8\_t \*u, char \*\*next, uint8\_t ulim)
- bool [strUInt8](#) (const char \*cs, uint8\_t \*u, char \*\*next)
- bool [strUInt](#) (const char \*cs, unsigned int \*u, char \*\*next)
- bool [strUInt16](#) (const char \*cs, uint16\_t \*u, char \*\*next)
- bool [strUInt32](#) (const char \*cs, uint32\_t \*u, char \*\*next)
- char \* [scaling](#) (float \*f, char \*s)
- bool [strFlt](#) (const char \*cs, float \*p, char \*\*next)
- unsigned int [strUInts](#) (const char \*cs, unsigned int \*p, unsigned int nmax, char \*\*next)
- unsigned int [strUInt32s](#) (const char \*cs, uint32\_t \*p, unsigned int nmax, char \*\*next)
- unsigned int [strFlts](#) (const char \*pc, float \*p, unsigned int nmax, char \*\*next)

## Variables

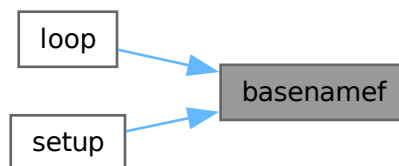
- char [printbuffer](#) [256]
- unsigned int [nprintbuffer](#) = 0

## 5.1.1 Function Documentation

### 5.1.1.1 basenamef()

```
char * basenamef (
    const char * cs)
```

Here is the caller graph for this function:



### 5.1.1.2 countWords()

```
unsigned int countWords (  
    char * s)
```

Here is the call graph for this function:



### 5.1.1.3 eos()

```
char * eos (  
    char * s)
```

### 5.1.1.4 eow()

```
char * eow (  
    char * s)
```

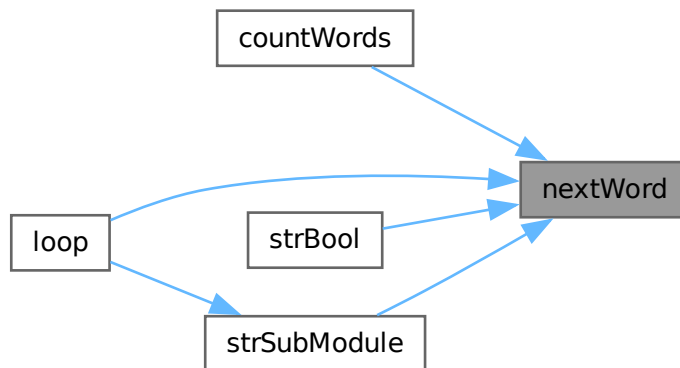
Here is the caller graph for this function:



#### 5.1.1.5 nextWord()

```
char * nextWord (  
    char * s)
```

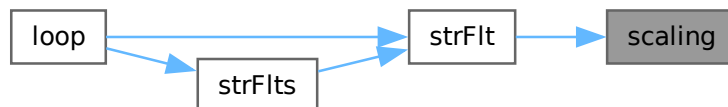
Here is the caller graph for this function:



#### 5.1.1.6 scaling()

```
char * scaling (  
    float * f,  
    char * s)
```

Here is the caller graph for this function:



#### 5.1.1.7 serialPrintf()

```
int serialPrintf (  
    const char * format,  
    ...)
```

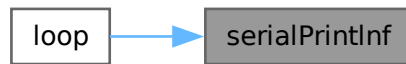
Here is the caller graph for this function:



#### 5.1.1.8 serialPrintlnf()

```
int serialPrintlnf (  
    const char * format,  
    ...)
```

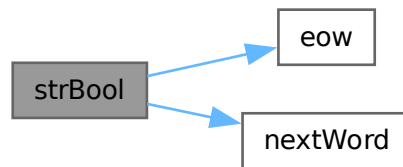
Here is the caller graph for this function:



#### 5.1.1.9 strBool()

```
bool strBool (  
    const char * cs,  
    bool * b,  
    char ** next)
```

Here is the call graph for this function:



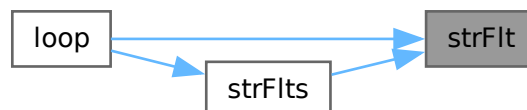
#### 5.1.1.10 strFlt()

```
bool strFlt (  
    const char * cs,  
    float * p,  
    char ** next)
```

Here is the call graph for this function:



Here is the caller graph for this function:





#### 5.1.1.11 strFlts()

```
unsigned int strFlts (  
    const char * pc,  
    float * p,  
    unsigned int nmax,  
    char ** next)
```

Here is the call graph for this function:



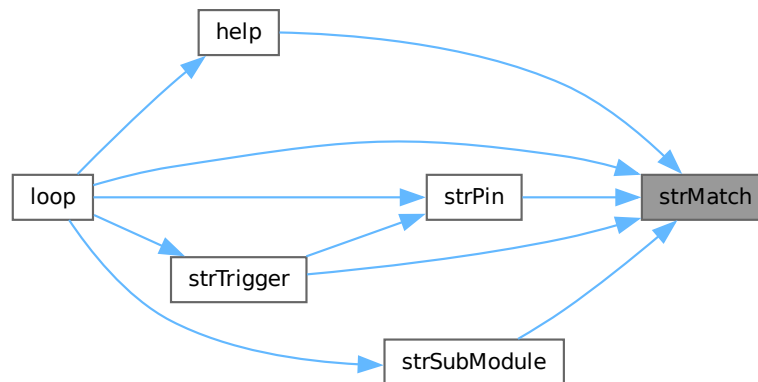
Here is the caller graph for this function:



#### 5.1.1.12 strMatch()

```
bool strMatch (  
    const char * cs,  
    const char * key,  
    char ** next)
```

Here is the caller graph for this function:



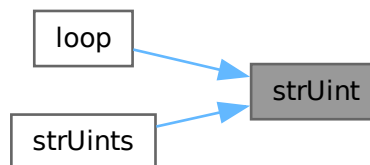
#### 5.1.1.13 strUInt()

```

bool strUInt (
    const char * cs,
    unsigned int * u,
    char ** next)

```

Here is the caller graph for this function:



#### 5.1.1.14 strUInt16()

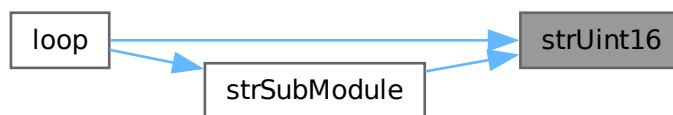
```

bool strUInt16 (
    const char * cs,

```

```
uint16_t * u,  
char ** next)
```

Here is the caller graph for this function:



#### 5.1.1.15 strUInt32()

```
bool strUInt32 (  
    const char * cs,  
    uint32_t * u,  
    char ** next)
```

Here is the caller graph for this function:



#### 5.1.1.16 strUInt32s()

```
unsigned int strUInt32s (  
    const char * cs,  
    uint32_t * p,  
    unsigned int nmax,  
    char ** next)
```

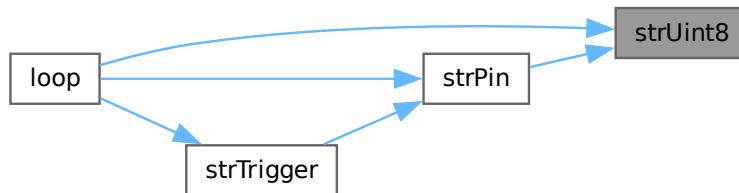
Here is the call graph for this function:



#### 5.1.1.17 strUint8()

```
bool strUint8 (  
    const char * cs,  
    uint8_t * u,  
    char ** next)
```

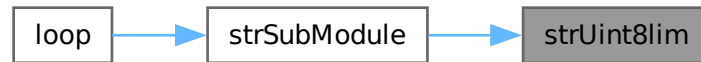
Here is the caller graph for this function:



#### 5.1.1.18 strUint8lim()

```
bool strUint8lim (  
    const char * cs,  
    uint8_t * u,  
    char ** next,  
    uint8_t ulim)
```

Here is the caller graph for this function:



#### 5.1.1.19 strUints()

```
unsigned int strUints (  
    const char * cs,  
    unsigned int * p,  
    unsigned int nmax,  
    char ** next)
```

Here is the call graph for this function:



#### 5.1.1.20 wordLength()

```
unsigned int wordLength (  
    char * s)
```

### 5.1.2 Variable Documentation

#### 5.1.2.1 nprintbuffer

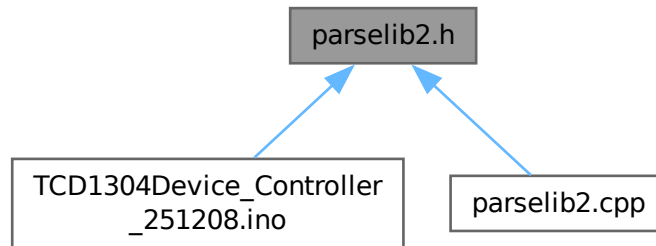
```
unsigned int nprintbuffer = 0
```

### 5.1.2.2 printbuffer

```
char printbuffer[256]
```

## 5.2 parselib2.h File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- `#define PARSELIB_H`

### Functions

- int `serialPrintf` (const char \*fmt,...)
- int `serialPrintlnf` (const char \*fmt,...)
- unsigned int `wordLength` (char \*s)
- char \* `nextWord` (char \*s)
- unsigned int `countWords` (char \*s)
- char \* `basenamef` (const char \*cs)
- bool `strMatch` (const char \*s, const char \*key, char \*\*next)
- bool `strBool` (const char \*s, bool \*b, char \*\*next=0)
- bool `strUInt8` (const char \*s, uint8\_t \*u, char \*\*next)
- bool `strUInt8lim` (const char \*s, uint8\_t \*u, char \*\*next, uint8\_t ulim)
- bool `strUInt` (const char \*s, unsigned int \*u, char \*\*next)
- bool `strUInt16` (const char \*s, uint16\_t \*u, char \*\*next)
- bool `strUInt32` (const char \*s, uint32\_t \*u, char \*\*next)
- bool `strFlt` (const char \*s, float \*p, char \*\*next)
- unsigned int `strUInts` (const char \*pc, unsigned int \*p, unsigned int nmax, char \*\*next)
- unsigned int `strUInt32s` (const char \*pc, uint32\_t \*p, unsigned int nmax, char \*\*next)
- unsigned int `strFlts` (const char \*pc, float \*p, unsigned int nmax, char \*\*next)

## 5.2.1 Macro Definition Documentation

### 5.2.1.1 PARSELIB\_H

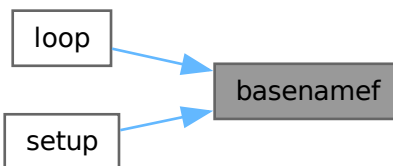
```
#define PARSELIB_H
```

## 5.2.2 Function Documentation

### 5.2.2.1 basenameef()

```
char * basenameef (  
    const char * cs)
```

Here is the caller graph for this function:



### 5.2.2.2 countWords()

```
unsigned int countWords (  
    char * s)
```

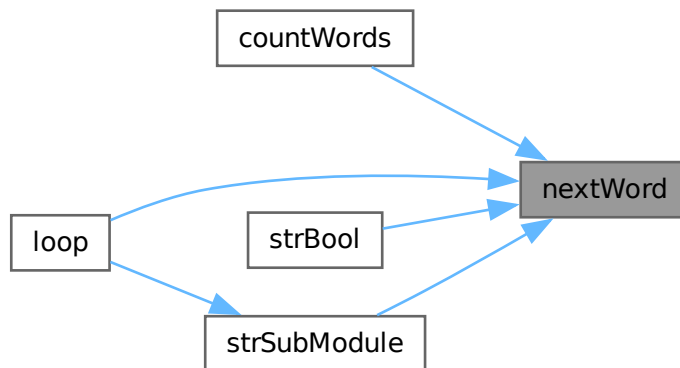
Here is the call graph for this function:



### 5.2.2.3 nextWord()

```
char * nextWord (  
    char * s)
```

Here is the caller graph for this function:



### 5.2.2.4 serialPrintf()

```
int serialPrintf (  
    const char * fmt,  
    ...)
```

Here is the caller graph for this function:





### 5.2.2.5 serialPrintlnf()

```
int serialPrintlnf (  
    const char * fmt,  
    ...)
```

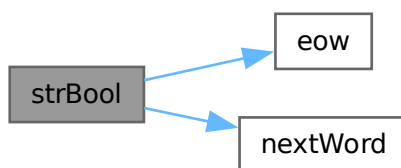
Here is the caller graph for this function:



### 5.2.2.6 strBool()

```
bool strBool (  
    const char * s,  
    bool * b,  
    char ** next = 0)
```

Here is the call graph for this function:



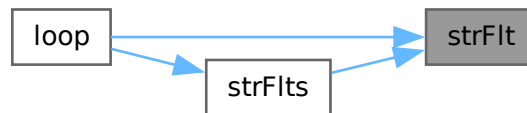
### 5.2.2.7 strFlt()

```
bool strFlt (  
    const char * s,  
    float * p,  
    char ** next)
```

Here is the call graph for this function:



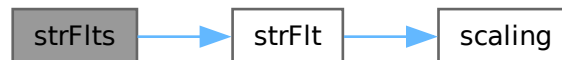
Here is the caller graph for this function:



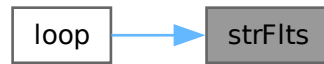
#### 5.2.2.8 strFlts()

```
unsigned int strFlts (  
    const char * pc,  
    float * p,  
    unsigned int nmax,  
    char ** next)
```

Here is the call graph for this function:



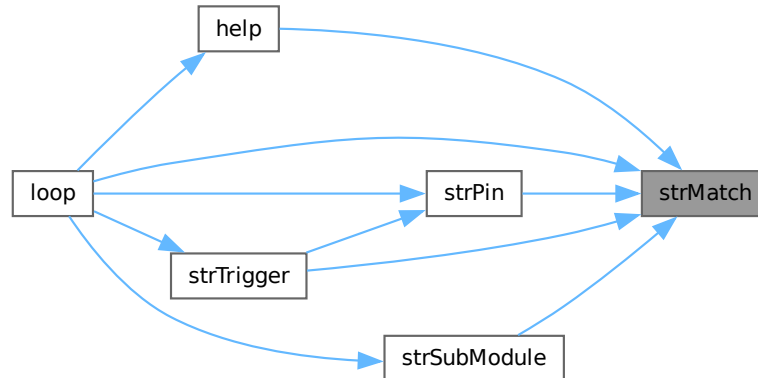
Here is the caller graph for this function:



### 5.2.2.9 strMatch()

```
bool strMatch (  
    const char * s,  
    const char * key,  
    char ** next)
```

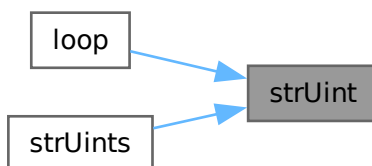
Here is the caller graph for this function:



### 5.2.2.10 strUInt()

```
bool strUInt (  
    const char * s,  
    unsigned int * u,  
    char ** next)
```

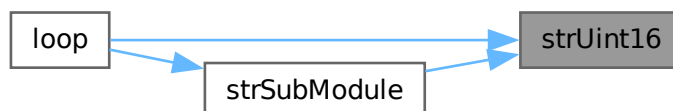
Here is the caller graph for this function:



#### 5.2.2.11 `strUInt16()`

```
bool strUInt16 (  
    const char * s,  
    uint16_t * u,  
    char ** next)
```

Here is the caller graph for this function:



#### 5.2.2.12 `strUInt32()`

```
bool strUInt32 (  
    const char * s,  
    uint32_t * u,  
    char ** next)
```

Here is the caller graph for this function:



#### 5.2.2.13 strUint32s()

```
unsigned int strUint32s (  
    const char * pc,  
    uint32_t * p,  
    unsigned int nmax,  
    char ** next)
```

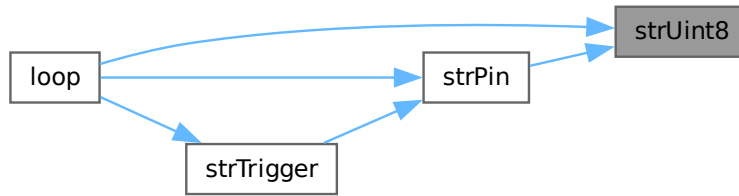
Here is the call graph for this function:



#### 5.2.2.14 strUint8()

```
bool strUint8 (  
    const char * s,  
    uint8_t * u,  
    char ** next)
```

Here is the caller graph for this function:



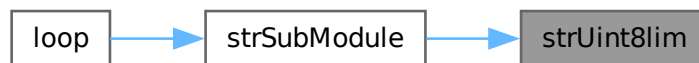
#### 5.2.2.15 strUInt8lim()

```

bool strUInt8lim (
    const char * s,
    uint8_t * u,
    char ** next,
    uint8_t ulim)

```

Here is the caller graph for this function:



#### 5.2.2.16 strUints()

```

unsigned int strUints (
    const char * pc,
    unsigned int * p,
    unsigned int nmax,
    char ** next)

```

Here is the call graph for this function:



### 5.2.2.17 wordLength()

```
unsigned int wordLength (
    char * s)
```

## 5.3 parselib2.h

[Go to the documentation of this file.](#)

```

00001 /* =====
00002     Parse strings for words, numbers, etc
00003
00004     Apart from wordLength(), these return a pointer to the
00005     next character in the string, or null if they fail
00006 */
00007 */
00008
00009 #ifndef PARSELIB_h
00010 #define PARSELIB_H
00011
00012 /*
00013 #ifdef __cplusplus
00014 extern "C" {
00015 #endif
00016 */
00017
00018 int serialPrintf( const char *fmt, ... );
00019 int serialPrintlnf( const char *fmt, ... );
00020
00021 unsigned int wordLength( char *s );
00022
00023 char *nextWord( char *s );
00024
00025 unsigned int countWords( char *s );
00026
00027 char *basenamef(const char *cs);
00028
00029 bool strMatch(const char *s, const char *key, char **next);
00030
00031 bool strBool(const char *s, bool *b, char **next=0);
00032
00033 bool strUInt8(const char *s, uint8_t *u, char **next );
00034
00035 bool strUInt8lim(const char *s, uint8_t *u, char **next, uint8_t ulim);
00036
00037 bool strUInt(const char *s, unsigned int *u, char **next);
00038
00039 bool strUInt16(const char *s, uint16_t *u, char **next);
00040
00041 bool strUInt32(const char *s, uint32_t *u, char **next);
00042

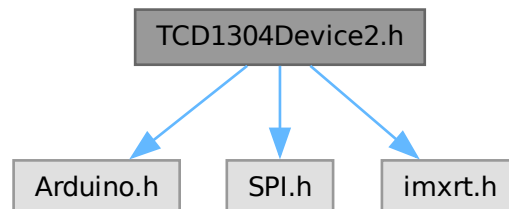
```

```
00043 bool strFlt(const char *s, float *p, char **next);
00044
00045 unsigned int strUints(const char *pc, unsigned int *p, unsigned int nmax, char **next);
00046
00047 unsigned int strUint32s(const char *pc, uint32_t *p, unsigned int nmax, char **next);
00048
00049 unsigned int strFlts(const char *pc, float *p, unsigned int nmax, char **next);
00050
00051 /*
00052  #ifdef __cplusplus
00053  }
00054  #endif
00055 */
00056
00057 #endif
```

## 5.4 README.md File Reference

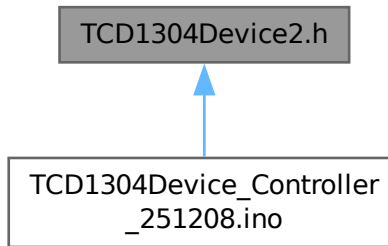
## 5.5 TCD1304Device2.h File Reference

```
#include "Arduino.h"
#include <SPI.h>
#include "imxrt.h"
Include dependency graph for TCD1304Device2.h:
```





This graph shows which files directly or indirectly include this file:



## Classes

- class [TCD1304Device](#)
- struct [TCD1304Device::SubModule](#)
- struct [TCD1304Device::Frame\\_Header\\_struct](#)

## Macros

- #define [PINPULLS](#)
- #define [TCD1304\\_MAXCLKHZ](#) 4.E6
- #define [TCD1304\\_MINCLKHZ](#) 0.8E6
- #define [TDIFF](#)(a, b)
- #define [SH\\_STOP\\_IN\\_READ](#)
- #define [SH\\_CLEARING\\_DEFAULT](#) 0
- #define [ROUNDUP](#)(a, b)
- #define [ROUNDO](#)(a, b)
- #define [ROUNDTOMOD](#)(a, b, c)
- #define [CYCCNT2SECS](#)(a)
- #define [PWM\\_CTRL2\\_CLOCK\\_MASTER](#) FLEXPWM\_SMCTRL2\_FRCEN
- #define [PWM\\_CTRL2\\_CLOCK\\_SLAVE](#) (FLEXPWM\_SMCTRL2\_CLK\_SEL(0x2) | (FLEXPWM\_SMCTRL2\_↵ FRCEN | FLEXPWM\_SMCTRL2\_FORCE\_SEL(0x3)))
- #define [PWM\\_CTRL2\\_CLOCK\\_SYNC](#) (FLEXPWM\_SMCTRL2\_CLK\_SEL(0x2) | FLEXPWM\_SMCTRL2\_↵ FRCEN)
- #define [CMPF\\_MASKA\\_ON](#) (1<<2)
- #define [CMPF\\_MASKA\\_OFF](#) (1<<3)
- #define [CMPF\\_MASKA\\_ON\\_OFF](#) (CMPF\_MASKA\_ON|CMPF\_MASKA\_OFF)
- #define [CMPF\\_MASKB\\_ON](#) (1<<4)
- #define [CMPF\\_MASKB\\_OFF](#) (1<<5)
- #define [CMPF\\_MASKB\\_ON\\_OFF](#) (CMPF\_MASKB\_ON|CMPF\_MASKB\_OFF)
- #define [CLK\\_DEFAULT](#) 64
- #define [CLK\\_SUBMODULE](#) 0
- #define [CLK\\_MASK](#) 0x1

- #define CLK\_CHANNEL 1
- #define CLK\_PIN 4
- #define CLK\_MUXVAL 1
- #define CLK\_CTRL2\_MASK PWM\_CTRL2\_CLOCK\_MASTER
- #define CLK\_IRQ IRQ\_FLEXPWM2\_0
- #define CLK\_CMPF\_MASK CMPF\_MASKA\_ON
- #define ICG\_SUBMODULE 1
- #define ICG\_MASK 0x2
- #define ICG\_CHANNEL 1
- #define ICG\_PIN 5
- #define ICG\_MUXVAL 1
- #define ICG\_CTRL2\_MASK PWM\_CTRL2\_CLOCK\_SLAVE
- #define ICG\_IRQ IRQ\_FLEXPWM2\_1
- #define ICG\_CMPF\_MASK (CMPF\_MASKA\_OFF|CMPF\_MASKB\_ON\_OFF)
- #define SH\_SUBMODULE 2
- #define SH\_MASK 0x4
- #define SH\_CHANNEL 1
- #define SH\_PIN 6
- #define SH\_MUXVAL 2
- #define SH\_CTRL2\_MASK PWM\_CTRL2\_CLOCK\_SLAVE
- #define SH\_IRQ IRQ\_FLEXPWM2\_2
- #define SH\_CMPF\_MASK CMPF\_MASKA\_ON\_OFF
- #define CNVST\_SUBMODULE 3
- #define CNVST\_MASK 0x8
- #define CNVST\_CHANNEL 1
- #define CNVST\_CTRL2\_MASK PWM\_CTRL2\_CLOCK\_SYNC
- #define CNVST\_IRQ IRQ\_FLEXPWM2\_3
- #define CNVST\_CMPF\_MASK CMPF\_MASKA\_OFF
- #define TIMER\_SUBMODULE 2
- #define TIMER\_MASK 0x4
- #define TIMER\_CHANNEL 2
- #define TIMER\_PIN 3
- #define TIMER\_MUXVAL 1
- #define TIMER\_CTRL2\_MASK PWM\_CTRL2\_CLOCK\_MASTER
- #define TIMER\_IRQ IRQ\_FLEXPWM4\_2
- #define TIMER\_CMPF\_MASK CMPF\_MASKB\_ON
- #define CNVST\_PIN 10
- #define SETCNVST (CORE\_PIN10\_PORTSET = CORE\_PIN10\_BITMASK)
- #define CNVST\_PULSE\_SECS 630.E-9
- #define CLEARCNVST (CORE\_PIN10\_PORTCLEAR = CORE\_PIN10\_BITMASK)
- #define CLK\_MONITOR\_PIN 3
- #define SYNC\_PIN 0
- #define BUSY\_PIN 1
- #define TRIGGER\_PIN 2
- #define SYNC\_PIN\_DEFAULT LOW
- #define BUSY\_PIN\_DEFAULT HIGH
- #define NREADOUT 3694
- #define DATASTART 16
- #define DATASTOP 3680
- #define NPIXELS (DATASTOP-DATASTART)
- #define NBYTES (NPIXELS\*2)

- `#define NBYTES32 (NPIXELS*4)`
- `#define NDARK 13`
- `#define NBITS 16`
- `#define VFS (0.6)`
- `#define VPERBIT (VFS/(1<<NBITS))`
- `#define SHUTTERMIN 5`
- `#define SETCNVST (CORE_PIN10_PORTSET = CORE_PIN10_BITMASK)`
- `#define CLEARCNVST (CORE_PIN10_PORTCLEAR = CORE_PIN10_BITMASK)`
- `#define USBSPEED 480.0E6`
- `#define USBTRANSFERSECS ((float)NREADOUT*(16/USBSPEED))`
- `#define COUNTER_MAX_SECS ((float)(65535 * 128) / F_BUS_ACTUAL)`
- `#define DEBUGPRINTF(...)`

## Enumerations

- enum `TCD1304_Mode_t` { `NOTCONFIGURED` , `PULSE` , `FRAMESET` , `TIMER` }

## 5.5.1 Macro Definition Documentation

### 5.5.1.1 BUSY\_PIN

```
#define BUSY_PIN 1
```

### 5.5.1.2 BUSY\_PIN\_DEFAULT

```
#define BUSY_PIN_DEFAULT HIGH
```

### 5.5.1.3 CLEARCNVST [1/2]

```
#define CLEARCNVST (CORE_PIN10_PORTCLEAR = CORE_PIN10_BITMASK)
```

### 5.5.1.4 CLEARCNVST [2/2]

```
#define CLEARCNVST (CORE_PIN10_PORTCLEAR = CORE_PIN10_BITMASK)
```

### 5.5.1.5 CLK\_CHANNEL

```
#define CLK_CHANNEL 1
```

### 5.5.1.6 CLK\_CMPF\_MASK

```
#define CLK_CMPF_MASK CMPF_MASKA_ON
```

#### 5.5.1.7 CLK\_CTRL2\_MASK

```
#define CLK_CTRL2_MASK PWM_CTRL2_CLOCK_MASTER
```

#### 5.5.1.8 CLK\_DEFAULT

```
#define CLK_DEFAULT 64
```

#### 5.5.1.9 CLK\_IRQ

```
#define CLK_IRQ IRQ_FLEXPWM2_0
```

#### 5.5.1.10 CLK\_MASK

```
#define CLK_MASK 0x1
```

#### 5.5.1.11 CLK\_MONITOR\_PIN

```
#define CLK_MONITOR_PIN 3
```

#### 5.5.1.12 CLK\_MUXVAL

```
#define CLK_MUXVAL 1
```

#### 5.5.1.13 CLK\_PIN

```
#define CLK_PIN 4
```

#### 5.5.1.14 CLK\_SUBMODULE

```
#define CLK_SUBMODULE 0
```

#### 5.5.1.15 CMPF\_MASKA\_OFF

```
#define CMPF_MASKA_OFF (1<<3)
```

#### 5.5.1.16 CMPF\_MASKA\_ON

```
#define CMPF_MASKA_ON (1<<2)
```

#### 5.5.1.17 CMPF\_MASKA\_ON\_OFF

```
#define CMPF_MASKA_ON_OFF (CMPF_MASKA_ON|CMPF_MASKA_OFF)
```

#### 5.5.1.18 CMPF\_MASKB\_OFF

```
#define CMPF_MASKB_OFF (1<<5)
```

#### 5.5.1.19 CMPF\_MASKB\_ON

```
#define CMPF_MASKB_ON (1<<4)
```

#### 5.5.1.20 CMPF\_MASKB\_ON\_OFF

```
#define CMPF_MASKB_ON_OFF (CMPF_MASKB_ON|CMPF_MASKB_OFF)
```

#### 5.5.1.21 CNVST\_CHANNEL

```
#define CNVST_CHANNEL 1
```

#### 5.5.1.22 CNVST\_CMPF\_MASK

```
#define CNVST_CMPF_MASK CMPF_MASKA_OFF
```

#### 5.5.1.23 CNVST\_CTRL2\_MASK

```
#define CNVST_CTRL2_MASK PWM_CTRL2_CLOCK_SYNC
```

#### 5.5.1.24 CNVST\_IRQ

```
#define CNVST_IRQ IRQ_FLEXPWM2_3
```

#### 5.5.1.25 CNVST\_MASK

```
#define CNVST_MASK 0x8
```

#### 5.5.1.26 CNVST\_PIN

```
#define CNVST_PIN 10
```

#### 5.5.1.27 CNVST\_PULSE\_SECS

```
#define CNVST_PULSE_SECS 630.E-9
```

#### 5.5.1.28 CNVST\_SUBMODULE

```
#define CNVST_SUBMODULE 3
```

#### 5.5.1.29 COUNTER\_MAX\_SECS

```
#define COUNTER_MAX_SECS ((float)(65535 * 128) / F_BUS_ACTUAL)
```

#### 5.5.1.30 CYCCNT2SECS

```
#define CYCCNT2SECS(  
    a)
```

**Value:**

```
((double) (a) / F_CPU)
```

#### 5.5.1.31 DATASTART

```
#define DATASTART 16
```

#### 5.5.1.32 DATASTOP

```
#define DATASTOP 3680
```

#### 5.5.1.33 DEBUGPRINTF

```
#define DEBUGPRINTF(  
    ...)
```

#### 5.5.1.34 ICG\_CHANNEL

```
#define ICG_CHANNEL 1
```

#### 5.5.1.35 ICG\_CMPF\_MASK

```
#define ICG_CMPF_MASK (CMPF_MASKA_OFF|CMPF_MASKB_ON_OFF)
```

#### 5.5.1.36 ICG\_CTRL2\_MASK

```
#define ICG_CTRL2_MASK PWM_CTRL2_CLOCK_SLAVE
```

#### 5.5.1.37 ICG\_IRQ

```
#define ICG_IRQ IRQ_FLEXPWM2_1
```

#### 5.5.1.38 ICG\_MASK

```
#define ICG_MASK 0x2
```

#### 5.5.1.39 ICG\_MUXVAL

```
#define ICG_MUXVAL 1
```

#### 5.5.1.40 ICG\_PIN

```
#define ICG_PIN 5
```

#### 5.5.1.41 ICG\_SUBMODULE

```
#define ICG_SUBMODULE 1
```

#### 5.5.1.42 NBITS

```
#define NBITS 16
```

#### 5.5.1.43 NBYTES

```
#define NBYTES (NPIXELS*2)
```

#### 5.5.1.44 NBYTES32

```
#define NBYTES32 (NPIXELS*4)
```

#### 5.5.1.45 NDARK

```
#define NDARK 13
```

#### 5.5.1.46 NPIXELS

```
#define NPIXELS (DATASTOP-DATASTART)
```

#### 5.5.1.47 NREADOUT

```
#define NREADOUT 3694
```

#### 5.5.1.48 PINPULLS

```
#define PINPULLS
```

#### 5.5.1.49 PWM\_CTRL2\_CLOCK\_MASTER

```
#define PWM_CTRL2_CLOCK_MASTER FLEXPWM_SMCTRL2_FRCEN
```

#### 5.5.1.50 PWM\_CTRL2\_CLOCK\_SLAVE

```
#define PWM_CTRL2_CLOCK_SLAVE (FLEXPWM_SMCTRL2_CLK_SEL(0x2) | (FLEXPWM_SMCTRL2_FRCEN | FLEXPWM_↔  
SMCTRL2_FORCE_SEL(0x3)))
```

#### 5.5.1.51 PWM\_CTRL2\_CLOCK\_SYNC

```
#define PWM_CTRL2_CLOCK_SYNC (FLEXPWM_SMCTRL2_CLK_SEL(0x2) | FLEXPWM_SMCTRL2_FRCEN)
```

#### 5.5.1.52 ROUNDTO

```
#define ROUNDTO(  
    a,  
    b)
```

**Value:**

$((a/b) * b)$

#### 5.5.1.53 ROUNDTOMOD

```
#define ROUNDTOMOD(  
    a,  
    b,  
    c)
```

**Value:**

$(( (a/b) * b) \% c)$



#### 5.5.1.54 ROUNDUP

```
#define ROUNDUP(  
    a,  
    b)
```

**Value:**

```
(ceil((float)a/b)*b)
```

#### 5.5.1.55 SETCNVST [1/2]

```
#define SETCNVST (CORE_PIN10_PORTSET = CORE_PIN10_BITMASK)
```

#### 5.5.1.56 SETCNVST [2/2]

```
#define SETCNVST (CORE_PIN10_PORTSET = CORE_PIN10_BITMASK)
```

#### 5.5.1.57 SH\_CHANNEL

```
#define SH_CHANNEL 1
```

#### 5.5.1.58 SH\_CLEARING\_DEFAULT

```
#define SH_CLEARING_DEFAULT 0
```

#### 5.5.1.59 SH\_CMPF\_MASK

```
#define SH_CMPF_MASK CMPF_MASKA_ON_OFF
```

#### 5.5.1.60 SH\_CTRL2\_MASK

```
#define SH_CTRL2_MASK PWM_CTRL2_CLOCK_SLAVE
```

#### 5.5.1.61 SH\_IRQ

```
#define SH_IRQ IRQ_FLEXPWM2_2
```

#### 5.5.1.62 SH\_MASK

```
#define SH_MASK 0x4
```

**5.5.1.63 SH\_MUXVAL**

```
#define SH_MUXVAL 2
```

**5.5.1.64 SH\_PIN**

```
#define SH_PIN 6
```

**5.5.1.65 SH\_STOP\_IN\_READ**

```
#define SH_STOP_IN_READ
```

**5.5.1.66 SH\_SUBMODULE**

```
#define SH_SUBMODULE 2
```

**5.5.1.67 SHUTTERMIN**

```
#define SHUTTERMIN 5
```

**5.5.1.68 SYNC\_PIN**

```
#define SYNC_PIN 0
```

**5.5.1.69 SYNC\_PIN\_DEFAULT**

```
#define SYNC_PIN_DEFAULT LOW
```

**5.5.1.70 TCD1304\_MAXCLKHZ**

```
#define TCD1304_MAXCLKHZ 4.E6
```

**5.5.1.71 TCD1304\_MINCLKHZ**

```
#define TCD1304_MINCLKHZ 0.8E6
```

#### 5.5.1.72 TDIFF

```
#define TDIFF(  
    a,  
    b)
```

**Value:**

```
((double) (a-b) / F_CPU)
```

#### 5.5.1.73 TIMER\_CHANNEL

```
#define TIMER_CHANNEL 2
```

#### 5.5.1.74 TIMER\_CMPF\_MASK

```
#define TIMER_CMPF_MASK CMPF_MASKB_ON
```

#### 5.5.1.75 TIMER\_CTRL2\_MASK

```
#define TIMER_CTRL2_MASK PWM_CTRL2_CLOCK_MASTER
```

#### 5.5.1.76 TIMER\_IRQ

```
#define TIMER_IRQ IRQ_FLEXPWM4_2
```

#### 5.5.1.77 TIMER\_MASK

```
#define TIMER_MASK 0x4
```

#### 5.5.1.78 TIMER\_MUXVAL

```
#define TIMER_MUXVAL 1
```

#### 5.5.1.79 TIMER\_PIN

```
#define TIMER_PIN 3
```

#### 5.5.1.80 TIMER\_SUBMODULE

```
#define TIMER_SUBMODULE 2
```

#### 5.5.1.81 TRIGGER\_PIN

```
#define TRIGGER_PIN 2
```

#### 5.5.1.82 USBSPEED

```
#define USBSPEED 480.0E6
```

#### 5.5.1.83 USBTRANSFERSECS

```
#define USBTRANSFERSECS ((float)NREADOUT*(16/USBSPEED))
```

#### 5.5.1.84 VFS

```
#define VFS (0.6)
```

#### 5.5.1.85 VPERBIT

```
#define VPERBIT (VFS/(1<<NBITS))
```

### 5.5.2 Enumeration Type Documentation

#### 5.5.2.1 TCD1304\_Mode\_t

```
enum TCD1304_Mode_t
```

Enumerator

NOTCONFIGURED	
PULSE	
FRAMESET	
TIMER	

## 5.6 TCD1304Device2.h

[Go to the documentation of this file.](#)

```

00001 /* tcd1304 flexpwm library
00002
00003     Author Mitchell C Nelson, PhD
00004     Copyright 2025
00005
00006     Free for non-commercial use.
00007
00008     No warranty and no representation of suitability for any purpose whatsoever
00009
00010 */
00011
00012 #ifndef TCD1304DEVICE_H
00013 #define TCD1304DEVICE_H
00014
00015 #include "Arduino.h"
00016 #include <SPI.h>
00017
00018 #include "imxrt.h"
00019
00020 // =====
00021 // All In One Board
00022
00023 // #define ALLINONEBOARD
00024
00025 // CONFIGURE PULLUPS/PULLDOWNS
00026 #define PINPULLS
00027
00028 // =====
00029
00030 #ifdef ALLINONEBOARD
00031 #include <ADC.h>
00032 #include <ADC_util.h>
00033 extern ADC *adc;
00034 #define ANALOGPIN A0
00035 #define TCD1304_MAXCLKHZ 2.35E6
00036 #else
00037 #define TCD1304_MAXCLKHZ 4.E6
00038 #endif
00039 #define TCD1304_MINCLKHZ 0.8E6
00040
00041 // =====
00042
00043 #define TDIFF(a,b) ((double)(a-b)/F_CPU)
00044
00045 // debug statements in setup
00046 // #define DEBUG
00047
00048 // we'll comment this out for the new boards
00049 #define SH_STOP_IN_READ
00050
00051 // clearing pulses, leave at 0 as default
00052 #define SH_CLEARING_DEFAULT 0
00053
00054 #define ROUNDUP(a,b) (ceil((float)a/b)*b)
00055 #define ROUNDT0(a,b) ((a/b)*b)
00056 #define ROUNDTOMOD(a,b,c) (((a/b)*b)%c)
00057
00058 // cpu cycle counter values to seconds
00059 // usage
00060 #ifndef CYCNT2SECS
00061 #define CYCNT2SECS(a) ((double)(a)/F_CPU)
00062 #endif
00063
00064 // Clock and force configurations
00065 #define PWM_CTRL2_CLOCK_MASTER FLEXPWM_SMCTRL2_FRCEN
00066 #define PWM_CTRL2_CLOCK_SLAVE (FLEXPWM_SMCTRL2_CLK_SEL(0x2) | (FLEXPWM_SMCTRL2_FRCEN |
    FLEXPWM_SMCTRL2_FORCE_SEL(0x3)))
00067 #define PWM_CTRL2_CLOCK_SYNC (FLEXPWM_SMCTRL2_CLK_SEL(0x2) | FLEXPWM_SMCTRL2_FRCEN)
00068
00069 // Flexpwm compare interrupts
00070 #define CMPF_MASKA_ON (1<<2)
00071 #define CMPF_MASKA_OFF (1<<3)
00072 #define CMPF_MASKA_ON_OFF (CMPF_MASKA_ON|CMPF_MASKA_OFF)
00073
00074 #define CMPF_MASKB_ON (1<<4)

```

```

00075 #define CMPF_MASKB_OFF (1<<5)
00076 #define CMPF_MASKB_ON_OFF (CMPF_MASKB_ON|CMPF_MASKB_OFF)
00077
00078 //#define CLK_DEFAULT 48
00079 #define CLK_DEFAULT 64
00080 //#define CLK_DEFAULT 96
00081 //#define CLK_DEFAULT 128
00082
00083 #define CLK_SUBMODULE 0
00084 #define CLK_MASK 0x1
00085 #define CLK_CHANNEL 1
00086 #define CLK_PIN 4
00087 #define CLK_MUXVAL 1
00088 #define CLK_CTRL2_MASK PWM_CTRL2_CLOCK_MASTER
00089 #define CLK_IRQ IRQ_FLEXPWM2_0
00090 #define CLK_CMPF_MASK CMPF_MASKA_ON
00091
00092 #define ICG_SUBMODULE 1
00093 #define ICG_MASK 0x2
00094 #define ICG_CHANNEL 1
00095 #define ICG_PIN 5
00096 #define ICG_MUXVAL 1
00097 #define ICG_CTRL2_MASK PWM_CTRL2_CLOCK_SLAVE
00098 #define ICG_IRQ IRQ_FLEXPWM2_1
00099 #define ICG_CMPF_MASK (CMPF_MASKA_OFF|CMPF_MASKB_ON_OFF)
00100
00101 #define SH_SUBMODULE 2
00102 #define SH_MASK 0x4
00103 #define SH_CHANNEL 1
00104 #define SH_PIN 6
00105 #define SH_MUXVAL 2
00106 #define SH_CTRL2_MASK PWM_CTRL2_CLOCK_SLAVE
00107 #define SH_IRQ IRQ_FLEXPWM2_2
00108 #define SH_CMPF_MASK CMPF_MASKA_ON_OFF
00109
00110 #define CNVST_SUBMODULE 3
00111 #define CNVST_MASK 0x8
00112 #define CNVST_CHANNEL 1
00113 #define CNVST_CTRL2_MASK PWM_CTRL2_CLOCK_SYNC
00114 #define CNVST_IRQ IRQ_FLEXPWM2_3
00115 #define CNVST_CMPF_MASK CMPF_MASKA_OFF
00116
00117 // this one is for PWM4
00118 #define TIMER_SUBMODULE 2
00119 #define TIMER_MASK 0x4
00120 #define TIMER_CHANNEL 2
00121 #define TIMER_PIN 3
00122 #define TIMER_MUXVAL 1
00123 #define TIMER_CTRL2_MASK PWM_CTRL2_CLOCK_MASTER
00124 #define TIMER_IRQ IRQ_FLEXPWM4_2
00125 #define TIMER_CMPF_MASK CMPF_MASKB_ON
00126
00127 /* Note that we do not configure the flexPWM to use this pin (CNVST_PIN),
00128    It is not on the mux list for this module,
00129    We operate this pin using digitalwrite, from the isr.
00130 */
00131 #define CNVST_PIN 10
00132 #define SETCNVST (CORE_PIN10_PORTSET = CORE_PIN10_BITMASK)
00133
00134 #define CNVST_PULSE_SECS 630.E-9
00135 //#define CNVST_OFFSET_CLOCKS 1
00136
00137 #define CLEARCNVST (CORE_PIN10_PORTCLEAR = CORE_PIN10_BITMASK)
00138
00139 #define CLK_MONITOR_PIN 3
00140
00141 #define SYNC_PIN 0
00142 #define BUSY_PIN 1
00143 #define TRIGGER_PIN 2
00144 //#define SPARE_PIN 3
00145
00146 #define SYNC_PIN_DEFAULT LOW
00147 #define BUSY_PIN_DEFAULT HIGH
00148
00149 // Fast nofrills BUSY pin set, lear, flip state
00150 /*
00151 #define SETBUSYPIN (CORE_PIN1_PORTSET = CORE_PIN1_BITMASK)
00152 #define CLEARBUSYPIN (CORE_PIN1_PORTCLEAR = CORE_PIN1_BITMASK)
00153 #define TOGGLEBUSYPIN (CORE_PIN1_PORTTOGGLE = CORE_PIN1_BITMASK)
00154 */
00155

```

```

00156 // Fast nofrills SYNC pin set/clear
00157 /*
00158 #define SETSYNCPIN (CORE_PIN0_PORTSET = CORE_PIN0_BITMASK)
00159 #define CLEARSYNCPIN (CORE_PIN0_PORTCLEAR = CORE_PIN0_BITMASK)
00160 #define TOGGLESYNCPIN (CORE_PIN0_PORTTOGGLE = CORE_PIN0_BITMASK)
00161 */
00162 // Sensor data readout
00163 #define NREADOUT 3694
00164 #define DATASTART 16
00165 #define DATASTOP 3680
00166
00167 // Size of the useful part
00168 #define NPIXELS (DATASTOP-DATASTART)
00169 #define NBYTES (NPIXELS*2)
00170 #define NBYTES32 (NPIXELS*4)
00171
00172 // And first part of that is dark
00173 #define NDARK 13
00174
00175 #ifdef ALLINONEBOARD
00176 #define NBITS 12
00177 #else
00178 #define NBITS 16
00179 #endif
00180
00181 #define VFS (0.6)
00182 #define VPERBIT (VFS/(1<<NBITS))
00183
00184 #define SHUTTERMIN 5
00185
00186 #define SETCNVST (CORE_PIN10_PORTSET = CORE_PIN10_BITMASK)
00187 #define CLEARCNVST (CORE_PIN10_PORTCLEAR = CORE_PIN10_BITMASK)
00188
00189 #define USBSPEED 480.0E6
00190 #define USBTRANSFERSECS ((float)NREADOUT*(16/USBSPEED))
00191
00192 #define COUNTER_MAX_SECS ((float)(65535 * 128) / F_BUS_ACTUAL)
00193
00194 #ifdef DEBUG
00195 int debugprintf(const char* format, ...)
00196 {
00197     char buffer[128] = {0};
00198     int n;
00199     va_list argptr;
00200     va_start(argptr, format);
00201     n = vsprintf (buffer, format, argptr );
00202     va_end(argptr);
00203     if (n > 0) {
00204         Serial.println(buffer);
00205     }
00206     else if (n<0) {
00207         Serial.print("Error: debugprintf vsprintf");
00208         Serial.println(format);
00209     }
00210     return n;
00211 }
00212 #define DEBUGPRINTF(...) debugprintf( __VA_ARGS__ )
00213 #else
00214 #define DEBUGPRINTF(...)
00215 #endif
00216
00217 // -----
00218 // NOTE this is not designed to be thread safe.
00219 // (Consider making it a singleton)
00220
00221 /*****
00222 * TCD1304 Device implemented in FlexPWM2, pins 3,4,5,6 and 10
00223 */
00224 typedef enum {
00225     NOTCONFIGURED,
00226     PULSE,
00227     FRAMESET,
00228     TIMER // we only have timer+pulse, not timer+frameset
00229 } TCD1304_Mode_t;
00230
00231 class TCD1304Device
00232 {
00233 public:
00234
00235     // struct to simplify our namespace for the flexpwm submodules
00236     struct SubModule {

```

```

00237     const char *name;
00238     const uint8_t submod;
00239     const uint8_t mask;
00240     const uint8_t pinA;
00241     const uint8_t muxvalA;
00242     const uint8_t pinB;
00243     const uint8_t muxvalB;
00244     IRQ_NUMBER_t irq;
00245     IMXRT_FLEXPWM_t *flexpwm;
00246     uint16_t period_counts = 0;
00247     uint16_t onA_counts = 0;
00248     uint16_t offA_counts = 0;
00249     uint16_t onB_counts = 0;
00250     uint16_t offB_counts = 0;
00251     uint16_t ctrl12_mask = 0;
00252     uint16_t intena_mask = 0;
00253     uint16_t divider = 1;
00254     uint8_t prescale = 0;
00255     uint8_t filler = 0;
00256     float period_secs = 0;
00257
00258     bool invertA = false;
00259     bool invertB = false;
00260
00261     bool newvals = false;
00262
00263     void (*isr)() = nullptr;
00264     uint16_t inten_mask = 0;
00265
00266     SubModule(const char *name_,
00267               uint8_t submod_, uint8_t mask_,
00268               uint8_t pinA_, uint8_t muxvalA_,
00269               uint8_t pinB_, uint8_t muxvalB_,
00270               IRQ_NUMBER_t irq_,
00271               IMXRT_FLEXPWM_t *flexpwm_) :
00272         name(name_),
00273         submod(submod_), mask(mask_),
00274         pinA(pinA_), muxvalA(muxvalA_),
00275         pinB(pinB_), muxvalB(muxvalB_),
00276         irq(irq_), flexpwm(flexpwm_) {};
00277 };
00278
00279 inline static uint64_t sh_cyccnt64_start = 0;
00280 inline static uint64_t sh_cyccnt64_now = 0;
00281 inline static uint64_t sh_cyccnt64_prev = 0;
00282 inline static uint64_t sh_cyccnt64_exposure = 0;
00283
00284 inline static uint64_t timer_cyccnt64_start = 0;
00285 inline static uint64_t timer_cyccnt64_now = 0;
00286 inline static uint64_t timer_cyccnt64_prev = 0;
00287
00288 // For the trigger timer
00289 inline static uint64_t trigger_cyccnt64_start = 0;
00290 inline static uint64_t trigger_cyccnt64_now = 0;
00291 inline static uint64_t trigger_cyccnt64_prev = 0;
00292
00293 // Referenced by timer and trigger setups and by the header routine
00294 inline static TCD1304_Mode_t mode = NOTCONFIGURED;
00295 inline static bool trigger_mode = false;
00296 inline static bool trigger_attached = false;
00297
00298 // bookkeeping counters
00299 inline static unsigned int sh_counter = 0;
00300 inline static unsigned int icg_counter = 0;
00301 inline static unsigned int cnvst_counter = 0;
00302 inline static unsigned int sh_counts_per_icg = 0;
00303
00304 // bookkeeping for the charge clearing pulses on SH
00305 inline static unsigned int sh_clearing_counts = SH_CLEARING_DEFAULT;
00306 inline static unsigned int sh_clearing_counter = 0;
00307 inline static unsigned int sh_short_period_counts = 0;
00308
00309 // read counters and callback
00310 inline static unsigned int read_counter = 0;
00311 inline static unsigned int read_counts = 0;
00312 inline static uint16_t *read_buffer = 0;
00313 inline static uint16_t *read_pointer = 0;
00314 inline static void (*read_callback)() = 0;
00315
00316 // frame counters and frameset callback
00317 inline static unsigned int frame_counter = 0;

```



```

00318 inline static unsigned int frame_counts = 0;
00319 inline static void (*frames_completed_callback)() = 0;
00320
00321 // framesets counters and framsets completed callback
00322 inline static unsigned int frameset_counter = 0;
00323 inline static unsigned int frameset_counts = 0;
00324 inline static void (*framesets_completed_callback)() = 0;
00325
00326 // timer stops at timer_outer_counts, timer_counter is for extended timer
00327 inline static unsigned int timer_inner_counter = 0;
00328 inline static unsigned int timer_inner_counts = 0;
00329 inline static unsigned int timer_outer_counter = 0;
00330 inline static unsigned int timer_outer_counts = 0;
00331 inline static void (*timer_callback)() = 0;
00332
00333 inline static float timer_interframe_min_secs = 0.;
00334 inline static float timer_period_secs = 0; // inner loop period
00335 inline static float timer_interval_secs = .0; // the resulting exposure interval
00336
00337 // interrupt (trigger) support
00338 inline static unsigned int trigger_counter = 0;
00339 inline static unsigned int trigger_counts = 1;
00340 inline static void (*trigger_callback)() = 0;
00341
00342 inline static uint8_t trigger_pin = TRIGGER_PIN;
00343 inline static uint8_t trigger_edge_mode = RISING;
00344 inline static uint8_t trigger_pin_mode = INPUT;
00345
00346 // timing adjustment
00347 #ifdef ALLINONEBOARD
00348 inline static uint16_t cnvst_extra_delay_counts = 0; // best signal, steadily decreases with
increasing delay
00349 #else
00350 inline static uint16_t cnvst_extra_delay_counts = 1; // this gives the lowest noise, does not improve
at 2
00351 #endif
00352
00353 // This is for frameset with exposure == frame interval, we skip the first icg for readout
00354 inline static bool skip_one = false;
00355 inline static bool skip_one_reload = false;
00356
00357 // State of the flexpwm interface
00358 inline static bool busytoggled = false;
00359
00360 // Sync pin management
00361 inline static uint8_t sync_pin = SYNC_PIN;
00362 inline static bool sync_toggled = false;
00363 inline static bool sync_enabled = true;
00364
00365 // And now.... the submodules
00366 inline static IMXRT_FLEXPWM_t * const flexpwm = &IMXRT_FLEXPWM2;
00367 inline static SubModule clk = {"clk", CLK_SUBMODULE, CLK_MASK, CLK_PIN, CLK_MUXVAL, 0xFF, 0, CLK_IRQ,
&IMXRT_FLEXPWM2};
00368 inline static SubModule sh = {"sh", SH_SUBMODULE, SH_MASK, SH_PIN, SH_MUXVAL, 0xFF, 0, SH_IRQ,
&IMXRT_FLEXPWM2};
00369 inline static SubModule icg = {"icg", ICG_SUBMODULE, ICG_MASK, ICG_PIN, ICG_MUXVAL, 0xFF, 0, ICG_IRQ,
&IMXRT_FLEXPWM2};
00370 inline static SubModule cnvst = {"cnvst", CNVST_SUBMODULE, CNVST_MASK, 0xFF, 0, 0xFF, 0, CNVST_IRQ,
&IMXRT_FLEXPWM2}; // no pin
00371
00372 // this is our interval clock, implemented on PWM4. option for pin3 output.
00373 inline static IMXRT_FLEXPWM_t * const timerflexpwm = &IMXRT_FLEXPWM4;
00374 inline static SubModule timer = {"timer", TIMER_SUBMODULE, TIMER_MASK, 0xFF, 0, TIMER_PIN, TIMER_MUXVAL,
TIMER_IRQ, &IMXRT_FLEXPWM4};
00375
00376 // error bookkeeping
00377 inline static bool error_flag = false;
00378 inline static bool oops_flag = false;
00379
00380
00381 //-----
00382 typedef struct Frame_Header_struct {
00383     uint16_t *buffer;
00384     unsigned int nbuffer;
00385
00386     // from the dummy outputs, first 16 elements
00387     unsigned int avgdummy;
00388     float offset;
00389
00390 #ifdef DEBUG
00391     uint64_t sh_cycnt64_start = 0;

```

```

00392     uint64_t sh_cyccnt64_now = 0;
00393     uint64_t sh_cyccnt64_prev = 0;
00394     uint64_t sh_cyccnt64_exposure = 0;
00395
00396     uint64_t timer_cyccnt64_start = 0;
00397     uint64_t timer_cyccnt64_now = 0;
00398     uint64_t timer_cyccnt64_prev = 0;
00399
00400     uint64_t trigger_cyccnt64_start = 0;
00401     uint64_t trigger_cyccnt64_now = 0;
00402     uint64_t trigger_cyccnt64_prev = 0;
00403
00404     float frame_difference_secs;
00405 #endif
00406
00407     float frame_elapsed_secs;
00408     float frame_exposure_secs;
00409
00410     float timer_elapsed_secs;
00411     float timer_difference_secs;
00412
00413     float trigger_elapsed_secs;
00414     float trigger_difference_secs;
00415
00416     unsigned int frame_counter;
00417     unsigned int frameset_counter;
00418     unsigned int trigger_counter;
00419
00420     TCD1304_Mode_t mode;
00421     bool trigger_mode;
00422
00423     bool error_flag;
00424     bool oops_flag;
00425
00426     bool frames_completed ;
00427     bool framesets_completed ;
00428
00429     bool ready_for_send;
00430 } Frame_Header;
00431
00432
00433 //-----
00434 TCD1304Device(unsigned int period=CLK_DEFAULT)
00435 {
00436     stop_with_irqs();
00437
00438     pinMode(trigger_pin, trigger_pin_mode);
00439
00440     pinMode(BUSY_PIN, OUTPUT);
00441     digitalWrite(BUSY_PIN, BUSY_PIN_DEFAULT);
00442
00443     pinMode(SYNC_PIN, OUTPUT);
00444     digitalWrite(SYNC_PIN, SYNC_PIN_DEFAULT);
00445
00446     pinMode(CNVST_PIN, OUTPUT);
00447     digitalWrite(CNVST_PIN, LOW);
00448
00449 #ifdef PINPULLS
00450     pinMode(CLK_PIN, INPUT_PULLDOWN);
00451     pinMode(SH_PIN, INPUT_PULLDOWN);
00452     pinMode(ICG_PIN, INPUT_PULLUP);
00453 #endif
00454
00455 }
00456
00457 void stop_all()
00458 {
00459     stop_with_irqs();
00460     clear_sync_busy_pins();
00461 }
00462
00463 static void stop_runs_only()
00464 {
00465     // Stops triggers
00466     stop_triggers();
00467
00468     // Stop all four submodules
00469     flexpwm_stop();
00470
00471     // Stop the timer
00472     timer_stop();

```

```

00473     }
00474
00475     static void disable_irqs()
00476     {
00477         // Disable all of the interrupts
00478         NVIC_DISABLE_IRQ(CLK_IRQ);
00479         NVIC_DISABLE_IRQ(ICG_IRQ);
00480         NVIC_DISABLE_IRQ(SH_IRQ);
00481         NVIC_DISABLE_IRQ(CNVST_IRQ);
00482
00483         // Disable the the timer interrupt
00484         NVIC_DISABLE_IRQ(TIMER_IRQ);
00485     }
00486
00487     static void stop_with_irqs()
00488     {
00489         // stop all of the flexpwms
00490         stop_runs_only();
00491
00492         // Disable all of the interrupts
00493         disable_irqs();
00494     }
00495
00496     static void clear_error_flags()
00497     {
00498         oops_flag = false;
00499         error_flag = false;
00500     }
00501
00502     static void clear_mode()
00503     {
00504         mode = NOTCONFIGURED;
00505         trigger_mode = false;
00506     }
00507
00508     static void clear_sync_busy_pins()
00509     {
00510         if (busytoggled) {
00511             busytoggled = false;
00512             digitalToggleFast(BUSY_PIN);
00513         }
00514
00515         // reset sync pin
00516         if (synctoggled) {
00517             digitalToggleFast(SYNC_PIN);
00518             synctoggled = false;
00519         }
00520     }
00521
00522     static void toggle_busypin()
00523     {
00524         busytoggled = !busytoggled;
00525         digitalToggleFast(BUSY_PIN);
00526     }
00527
00528     static void toggle_syncpin()
00529     {
00530         synctoggled = !synctoggled;
00531         digitalToggleFast(SYNC_PIN);
00532     }
00533
00534     static void clear_busypin()
00535     {
00536         if (busytoggled) {
00537             busytoggled = false;
00538             digitalToggleFast(BUSY_PIN);
00539         }
00540     }
00541
00542     static void clear_syncpin()
00543     {
00544         if (synctoggled) {
00545             synctoggled = false;
00546             digitalToggleFast(SYNC_PIN);
00547         }
00548     }
00549
00550     /* =====
00551     For ring buffering, we'll use this in the read callback to point to the next buffer
00552     */
00553     static void update_read_buffer(uint16_t *buffer)

```

```

00554 {
00555     read_buffer = buffer;
00556     //read_counts = nbuffer;
00557
00558     read_pointer = read_buffer;
00559     read_counter = 0;
00560 }
00561
00562 static void fill_frame_header(Frame_Header *p)
00563 {
00564     unsigned int utmp = 0;
00565     int n;
00566
00567     p->buffer = read_buffer;
00568     p->nbuffer = read_counts;
00569
00570     for (n=0; n<DATASTART; n++) {
00571         utmp += read_buffer[n];
00572     }
00573     p->avgdummy = utmp/DATASTART;
00574     p->offset = ((float)utmp * VPERBIT)/DATASTART;
00575
00576 #ifdef DEBUG
00577     p->sh_cyccnt64_start = sh_cyccnt64_start;
00578     p->sh_cyccnt64_now = sh_cyccnt64_now;
00579     p->sh_cyccnt64_prev = sh_cyccnt64_prev;
00580     p->sh_cyccnt64_exposure = sh_cyccnt64_exposure;
00581
00582     p->timer_cyccnt64_start = timer_cyccnt64_start;
00583     p->timer_cyccnt64_now = timer_cyccnt64_now;
00584     p->timer_cyccnt64_prev = timer_cyccnt64_prev;
00585
00586     p->trigger_cyccnt64_start = trigger_cyccnt64_start;
00587     p->trigger_cyccnt64_now = trigger_cyccnt64_now;
00588     p->trigger_cyccnt64_prev = trigger_cyccnt64_prev;
00589
00590     p->frame_difference_secs = sh_difference_secs();
00591 #endif
00592
00593     p->frame_counter = frame_counter;
00594     p->frameset_counter = frameset_counter;
00595
00596     p->trigger_counter = trigger_counter;
00597
00598     p->frames_completed = (frame_counter+1) == frame_counts ? true : false;
00599     p->framesets_completed = p->frames_completed && ((frameset_counter+1) == frameset_counts) ? true :
false;
00600
00601     p->frame_elapsed_secs = sh_elapsed_secs();
00602     p->frame_exposure_secs = sh_exposure_secs();
00603
00604     p->timer_elapsed_secs = timer_elapsed_secs();
00605     p->timer_difference_secs = timer_difference_secs();
00606
00607     p->trigger_elapsed_secs = trigger_elapsed_secs();
00608     p->trigger_difference_secs = trigger_difference_secs();
00609
00610     p->mode = mode;
00611     p->trigger_mode = trigger_mode;
00612
00613     p->error_flag = error_flag;
00614     p->oops_flag = oops_flag;
00615
00616     p->ready_for_send = true;
00617 }
00618
00619 /* =====
00620     High level API
00621 */
00622
00623 inline static float read_expected_time = 0.;
00624
00625 bool read(uint nframes, float exposure, uint16_t *bufferp,
00626          void (*frame_callbackf)(),
00627          void (*frameset_callbackf)(),
00628          void (*completion_callbackf)(),
00629          void (*setup_callbackf)(),
00630          bool start=true)
00631 {
00632     float clk_secs = 0.5E-6;
00633     float sh_secs = 1.0E-6;

```

```
00634     float sh_offset_secs = 0.6E-6;
00635     float icg_secs = 2.6E-6;
00636     float icg_offset_secs = 0.5E-6;
00637
00638     if (!setup_pulse(clk_secs, sh_secs, sh_offset_secs, icg_secs, icg_offset_secs,
00639                     bufferp, NREADOUT, frame_callbackf)) {
00640         Serial.println("Error: failed to setup pulse");
00641         return false;
00642     }
00643
00644     frame_counts = nframes;
00645     frames_completed_callback = frameset_callbackf;
00646     framesets_completed_callback = completion_callbackf;
00647
00648     if (!setup_timer(exposure, 0., nframes)) {
00649         Serial.println("Error: failed to setup timer");
00650         return false;
00651     }
00652
00653     if (setup_callbackf) {
00654         setup_callbackf();
00655     }
00656
00657     if (!start) {
00658         return true;
00659     }
00660
00661     // start the reads here
00662     timer_start();
00663     if (error_flag) {
00664         return false;
00665     }
00666
00667     read_expected_time = exposure * nframes;
00668
00669     return true;
00670 }
00671
00672 bool read(uint nframes, float exposure, float frame_interval, uint16_t *bufferp,
00673          void (*frame_callbackf)(),
00674          void (*frameset_callbackf)(),
00675          void (*completion_callbackf)(),
00676          void (*setup_callbackf)(),
00677          bool start=true)
00678 {
00679
00680     float clk_secs = 0.5E-6;
00681     float sh_secs = 1.0E-6;
00682     float sh_offset_secs = 0.6E-6;
00683     float icg_secs = 2.6E-6;
00684     float icg_offset_secs = 0.5E-6;
00685
00686     if (!setup_frameset(clk_secs, sh_secs, sh_offset_secs, icg_secs, icg_offset_secs,
00687                        exposure, frame_interval, nframes,
00688                        bufferp, NREADOUT, frame_callbackf)) {
00689         Serial.println("Error: failed to setup frameset");
00690         return false;
00691     }
00692
00693     frames_completed_callback = frameset_callbackf;
00694     framesets_completed_callback = completion_callbackf;
00695
00696
00697     if (setup_callbackf) {
00698         setup_callbackf();
00699     }
00700
00701
00702     if (!start) {
00703         return true;
00704     }
00705
00706     // start the reads here
00707     frameset_start();
00708     if (error_flag) {
00709         return false;
00710     }
00711
00712     read_expected_time = frame_interval * nframes;
00713
00714     return true;
```

```

00715     }
00716
00717     bool start_read()
00718     {
00719         if (mode == TIMER) {
00720             timer_start();
00721             return !error_flag;
00722         }
00723         else if (mode == FRAMESET) {
00724             frameset_start();
00725             return !error_flag;
00726         }
00727
00728         Serial.println("Error: start_read, not configured");
00729         return false;
00730     }
00731
00732     bool wait_read(float timeout=0., float timestep=0.01, bool verbose=false)
00733     {
00734         if (!timeout) timeout = read_expected_time - timer_elapsed_secs() + 0.010;
00735         if (timeout < 0.1) timeout = 0.1;
00736
00737         if (mode == TIMER) {
00738             return timer_wait(timeout,timestep, verbose);
00739         }
00740         else if (mode == FRAMESET) {
00741             return flexpwm_wait(timeout,timestep, verbose);
00742         }
00743
00744         Serial.println("Error: wait_read not configured");
00745         return false;
00746     }
00747
00748     // -----
00749     bool triggered_read(uint ntriggers, uint nframes, float exposure, uint16_t *bufferp,
00750                        void (*frame_callbackf)(),
00751                        void (*frameset_callbackf)(),
00752                        void (*completion_callbackf)(),
00753                        void (*setup_callbackf)(),
00754                        bool start=true)
00755     {
00756         if (!read(nframes, exposure, bufferp, frame_callbackf, frameset_callbackf, completion_callbackf,
00757                 nullptr, false)) {
00758             Serial.println("Error: triggerd_read setup read");
00759             return false;
00760         }
00761         if (!setup_triggers(ntriggers)) {
00762             Serial.println("Error: triggerd_read setup triggers");
00763             return false;
00764         }
00765
00766         if (setup_callbackf) setup_callbackf();
00767
00768         if (start) {
00769             return start_triggers();
00770         }
00771
00772         return true;
00773     }
00774
00775     bool triggered_read(uint ntriggers, uint nframes, float exposure, float interval, uint16_t *bufferp,
00776                        void (*frame_callbackf)(),
00777                        void (*frameset_callbackf)(),
00778                        void (*completion_callbackf)(),
00779                        void (*setup_callbackf)(),
00780                        bool start=true)
00781     {
00782         if (!read(nframes, exposure, interval, bufferp,
00783                 frame_callbackf, frameset_callbackf, completion_callbackf, nullptr, false)) {
00784             Serial.println("Error: triggerd_read setup read");
00785             return false;
00786         }
00787
00788         if (!setup_triggers(ntriggers)) {
00789             Serial.println("Error: triggerd_read setup trigger");
00790             return false;
00791         }
00792
00793         if (setup_callbackf) setup_callbackf();
00794

```

```

00795     if (start) {
00796         return start_triggers();
00797     }
00798
00799     return true;
00800 }
00801
00802 bool wait_triggered_read(float timeout=1., float timestep=0.01, bool verbose=false)
00803 {
00804     if (wait_triggers(timeout,timestep,verbose)) {
00805         return wait_read(timeout,timestep,verbose);
00806     }
00807
00808     return false;
00809 }
00810
00811
00812 /* =====
00813     64 bit elapsed time clock based on cpu cycles
00814     note that this is in the tcd1304 namespace
00815 */
00816 static uint64_t cycles64()
00817 {
00818     static uint32_t oldCycles = ARM_DWT_CYCCNT;
00819     static uint32_t highDWORD = 0;
00820     uint32_t newCycles = ARM_DWT_CYCCNT;
00821     if (newCycles < oldCycles)
00822     {
00823         ++highDWORD;
00824     }
00825     oldCycles = newCycles;
00826     return (((uint64_t)highDWORD << 32) | newCycles);
00827 }
00828
00829 /* -----
00830     Elapsed time from first frame (sh) to most recent (cf actual timer interval)
00831 */
00832 static double sh_elapsed_secs()
00833 {
00834     return (double)(sh_cyccnt64_now - sh_cyccnt64_start)/F_CPU;
00835 }
00836
00837 static double sh_difference_secs()
00838 {
00839     return (double)(sh_cyccnt64_now - sh_cyccnt64_prev)/F_CPU;
00840 }
00841
00842 static double sh_exposure_secs()
00843 {
00844     return (double)sh_cyccnt64_exposure/F_CPU;
00845 }
00846 /* -----
00847     Elapsed time from timer start to most recent timer interrupt
00848 */
00849 static double timer_elapsed_secs()
00850 {
00851     return (double)(timer_cyccnt64_now - timer_cyccnt64_start)/F_CPU;
00852 }
00853
00854 static double timer_difference_secs()
00855 {
00856     return (double)(timer_cyccnt64_now - timer_cyccnt64_prev)/F_CPU;
00857 }
00858
00859 /* -----
00860     Elapsed time from timer start to most recent timer interrupt
00861 */
00862 static double trigger_elapsed_secs()
00863 {
00864     return (double)(trigger_cyccnt64_now - trigger_cyccnt64_start)/F_CPU;
00865 }
00866
00867 static double trigger_difference_secs()
00868 {
00869     return (double)(trigger_cyccnt64_now - trigger_cyccnt64_prev)/F_CPU;
00870 }
00871
00872 /* =====
00873     Print error message
00874 */
00875 static void print_errormsg(const char *name, const char *errmsg)

```

```

00876 {
00877     Serial.print("Error: ");
00878     Serial.print(name);
00879     Serial.print(" ");
00880     Serial.println(errmsg);
00881 }
00882
00883 static void print_errormsg(SubModule *p, const char *errmsg)
00884 {
00885     print_errormsg(p->name,errmsg);
00886 }
00887
00888 static void print_counters()
00889 {
00890     Serial.print(" sh "); Serial.print(sh_counter);
00891     Serial.print(" icg "); Serial.print(icg_counter);
00892     Serial.print(" cnvst "); Serial.print(cnvst_counter);
00893     Serial.print(" read "); Serial.print(read_counter);
00894     Serial.print(" "); Serial.print(read_counts);
00895     Serial.print(" frame "); Serial.print(frame_counter);
00896     Serial.print(" "); Serial.print(frame_counts);
00897     Serial.print(" frameset "); Serial.print(frameset_counter);
00898     Serial.print(" "); Serial.print(frameset_counts);
00899     if (mode==TIMER) {
00900         Serial.print(" timer "); Serial.print(timer_inner_counter);
00901         Serial.print(" "); Serial.print(timer_inner_counts);
00902         Serial.print(" outer "); Serial.print(timer_outer_counter);
00903         Serial.print(" "); Serial.print(timer_outer_counts);
00904     }
00905     if (trigger_mode) {
00906         Serial.print(" triggers "); Serial.print(trigger_counter);
00907         Serial.print(" "); Serial.print(trigger_counts);
00908     }
00909     Serial.println("");
00910 }
00911
00912 /* =====
00913    Print, check, setup submodule configuration, load into hardware
00914 */
00915 void print_submodule(SubModule *p)
00916 {
00917     char cbuffer[128] = {0};
00918
00919     float frequency = p->period_counts ? F_BUS_ACTUAL/(p->divider*p->period_counts) : 0;
00920     float period_secs = (float) (p->period_counts*p->divider)/F_BUS_ACTUAL;
00921
00922     float A_on_secs = (float) (p->onA_counts*p->divider)/F_BUS_ACTUAL;
00923     float A_off_secs = (float) (p->offA_counts*p->divider)/F_BUS_ACTUAL;
00924     float A_duration_secs = (float) ((p->offA_counts-p->onA_counts)*p->divider)/F_BUS_ACTUAL;
00925
00926     float B_on_secs = (float) (p->onB_counts*p->divider)/F_BUS_ACTUAL;
00927     float B_off_secs = (float) (p->offB_counts*p->divider)/F_BUS_ACTUAL;
00928     float B_duration_secs = (float) ((p->offB_counts-p->onB_counts)*p->divider)/F_BUS_ACTUAL;
00929
00930     snprintf( cbuffer, sizeof(cbuffer),
00931         "FLEXPWM: %s submod %d mask 0x%x pinA %d muxA %d pinB %d muxB %d irq %d flexpwm %p",
00932         p->name, p->submod, p->mask, p->pinA, p->muxvalA, p->pinB, p->muxvalB, p->irq, p->flexpwm);
00933     Serial.println(cbuffer);
00934
00935     snprintf( cbuffer, sizeof(cbuffer),
00936         "flexpwm: %s period %u presc %d div %d => %.6g secs %.6g Hz ctrl2_mask x%02x",
00937         p->name, p->period_counts, p->prescale, p->divider, period_secs, frequency, p->ctrl2_mask);
00938     Serial.println(cbuffer);
00939
00940     snprintf( cbuffer, sizeof(cbuffer),
00941         "flexpwm: %s A pin %d mux 0x%x on %u off %u %s => %.6g to %.6g, %.6g secs",
00942         p->name, p->pinA, p->muxvalA, p->onA_counts, p->offA_counts,
00943         p->invertA?"inverting":"noninverting",
00944         A_on_secs, A_off_secs, A_duration_secs);
00945     Serial.println(cbuffer);
00946
00947     snprintf( cbuffer, sizeof(cbuffer),
00948         "flexpwm: %s B pin %d mux 0x%x on %u off %u %s => %.6g to %.6g, %.6g secs",
00949         p->name, p->pinB, p->muxvalB, p->onB_counts, p->offB_counts,
00950         p->invertB?"inverting":"noninverting",
00951         B_on_secs, B_off_secs, B_duration_secs);
00952     Serial.println(cbuffer);
00953 }
00954
00955 // -----
00956

```



```

00957 bool check_submodule(SubModule *p)
00958 {
00959     bool retv = true;
00960     if (p->divider<1) {
00961         print_errormsg(p->name, "divider < 1");
00962         retv = false;
00963     }
00964     if (p->divider>128) {
00965         print_errormsg(p->name, "divider > 128");
00966         retv = false;
00967     }
00968
00969     if (p->onA_counts || p->offA_counts) {
00970         if (p->onA_counts>=p->offA_counts) {
00971             print_errormsg(p->name, "onA > offA");
00972             retv = false;
00973         }
00974         if (p->offA_counts>p->period_counts) {
00975             print_errormsg(p->name, "offA >= period");
00976             retv = false;
00977         }
00978     }
00979
00980     if (p->onB_counts || p->offB_counts) {
00981         if (p->onB_counts>=p->offB_counts) {
00982             print_errormsg(p->name, "onB > offB");
00983             retv = false;
00984         }
00985         if (p->offB_counts>p->period_counts) {
00986             print_errormsg(p->name, "offB >= period");
00987             retv = false;
00988         }
00989     }
00990
00991     return retv;
00992 }
00993
00994 // -----
00995 bool print_and_check_submodule(SubModule *p)
00996 {
00997     print_submodule(p);
00998     return check_submodule(p);
00999 }
01000
01001 // -----
01002 void load_submodule(SubModule *p)
01003 {
01004     IMXRT_FLEXPWM_t *q = p->flexpwm;
01005     unsigned int submod = p->submod;
01006     uint16_t mask = p->mask;
01007
01008     Serial.print("loading sub_module ");
01009     Serial.println(p->name);
01010
01011     // stop this channel - harmless if it was done globally already
01012     q->MCTRL |= FLEXPWM_MCTRL_CLDOK(mask);
01013     q->MCTRL &= ~FLEXPWM_MCTRL_RUN(mask);
01014
01015     q->SM[submod].CTRL = FLEXPWM_SMCTRL_FULL | FLEXPWM_SMCTRL_PRSC(p->prescale);
01016
01017     q->SM[submod].INIT = 0;
01018     q->SM[submod].VAL0 = 0;
01019     q->SM[submod].VAL1 = p->period_counts - 1;
01020     q->SM[submod].VAL2 = p->onA_counts;
01021     q->SM[submod].VAL3 = (p->offA_counts > 0) ? p->offA_counts : 0;
01022     q->SM[submod].VAL4 = p->onB_counts;
01023     q->SM[submod].VAL5 = (p->offB_counts > 0) ? p->offB_counts : 0;
01024
01025     // Convenience, save the clock period in seconds
01026     p->period_secs = (float) p->period_counts * ((float) (1<p->prescale) / F_BUS_ACTUAL);
01027
01028     // Do we have a Pin for the A channel?
01029     if (p->pinA != 0xFF && p->offA_counts) {
01030
01031         q->OUTEN |= FLEXPWM_OUTEN_PWMA_EN(mask);
01032
01033         *(portConfigRegister(p->pinA)) = p->muxvalA;
01034
01035         if (p->invertA) {
01036             q->SM[submod].OCTRL |= 1<10; // Is inverted
01037         }

```

```

01038         else {
01039             q->SM[submod].OCTRL &= ~(1<10); // Is not inverted
01040         }
01041     }
01042     else {
01043         q->OUTEN &= ~FLEXPWM_OUTEN_PWMA_EN(mask);
01044     }
01045
01046     // Do we have a Pin for the B channel?
01047     if (p->pinB != 0xFF && p->offB_counts) {
01048
01049         q->OUTEN |= FLEXPWM_OUTEN_PWMB_EN(mask);
01050
01051         *(portConfigRegister(p->pinB)) = p->muxvalB;
01052
01053         if (p->invertB) {
01054             q->SM[submod].OCTRL |= 1<9; // Is inverted
01055         }
01056         else {
01057             q->SM[submod].OCTRL &= ~(1<9); // Is not inverted
01058         }
01059     }
01060     else {
01061         q->OUTEN &= ~FLEXPWM_OUTEN_PWMB_EN(mask);
01062     }
01063
01064     // Setup shared clocks and forced starts
01065     q->SM[submod].CTRL2 = p->ctrl2_mask;
01066
01067     p->newvals = false;
01068 }
01069
01070 // -----
01071 bool setup_submodule( SubModule *p, uint8_t prescale, uint16_t period_counts,
01072                     uint16_t onA_counts, uint16_t offA_counts, bool invertA,
01073                     uint16_t onB_counts, uint16_t offB_counts, bool invertB,
01074                     uint16_t ctrl2_mask)
01075 {
01076     if (p) {
01077         p->divider = (1<p->prescale);
01078         p->period_counts = period_counts;
01079
01080         p->onA_counts = onA_counts;
01081         p->offA_counts = offA_counts;
01082         p->invertA = invertA;
01083
01084         p->onB_counts = onB_counts;
01085         p->offB_counts = offB_counts;
01086         p->invertB = invertB;
01087
01088         p->ctrl2_mask = ctrl2_mask;
01089
01090         if (print_and_check_submodule(p)) {
01091             load_submodule(p);
01092             return true;
01093         }
01094     }
01095     return false;
01096 }
01097
01098 /* =====
01099     Attach ISR and enable
01100     see above, CMPF_MASKA_ON, CMPF_MASKA_OFF, etc
01101 */
01102 void attach_isr( SubModule *p, uint16_t cmpf_mask, void (*isrf)())
01103 {
01104     IMXRT_FLEXPWM_t *q = p->flexpwm;
01105     unsigned int submod = p->submod;
01106     uint16_t status;
01107
01108     // bookkeeping to support renabling
01109     p->inten_mask = cmpf_mask;
01110     p->isr = isrf;
01111
01112     // disable the irq for this submodule
01113     NVIC_DISABLE_IRQ(p->irq);
01114
01115     // clear all of this module's interrupt status bits
01116     status = q->SM[submod].STS;
01117     q->SM[submod].STS = status;
01118 }

```

```

01119 // enable the specified bits only
01120 q->SM[submod].INTEN = cmpf_mask;
01121 p->intena_mask = cmpf_mask;
01122
01123 // register the isr to this irq
01124 attachInterruptVector(p->irq, isrf);
01125
01126 // and now, enable interrupts on this irq
01127 NVIC_ENABLE_IRQ(p->irq);
01128 }
01129
01130
01131 /* =====
01132 Setup callbacks for frames (a single frameset) completed.
01133 */
01134 void clear_frames_completed_callback()
01135 {
01136     frames_completed_callback = 0;
01137 }
01138
01139 void load_frames_completed_callback(void (*callback)(), unsigned int nframes=0)
01140 {
01141     // allow for rearm callback. same nframes
01142     if (nframes) {
01143         frame_counts = nframes;
01144     }
01145     frames_completed_callback = callback;
01146 }
01147
01148 /* =====
01149 Setup callbacks for frameset completed.
01150 */
01151 void clear_framesets_completed_callback()
01152 {
01153     framesets_completed_callback = 0;
01154 }
01155
01156 void load_framesets_completed_callback(void (*callback)(), unsigned int nsets=0)
01157 {
01158     // allow for rearm callback. same nframes
01159     if (nsets) {
01160         frameset_counts = nsets;
01161     }
01162     framesets_completed_callback = callback;
01163 }
01164
01165 /* =====
01166 FlexPWM stop and wait
01167 */
01168
01169 inline static bool flexpwm_running = false;
01170
01171 static void flexpwm_start()
01172 {
01173     oops_flag = false;
01174     error_flag = false;
01175     flexpwm_running = true;
01176     flexpwm->MCTRL |= 0xF; // set load ok for all four submodules
01177     flexpwm->MCTRL |= ((CLK_MASK|SH_MASK|ICG_MASK) << 8); // set run clk, sh, icg
01178 }
01179
01180 static void flexpwm_stop()
01181 {
01182     flexpwm->MCTRL |= FLEXPWM_MCTRL_CLDOK(0xF);
01183     flexpwm->MCTRL = 0; // stop everything
01184     flexpwm_running = false;
01185 }
01186
01187 bool flexpwm_wait(float timeout_=1., float timestep_=0.01, bool verbose=false)
01188 {
01189     uint32_t timeout = (uint) (timeout_ * 1000);
01190     uint32_t increment = (uint) (timestep_ * 1000);
01191     uint32_t elapsed = 0;
01192
01193     if (verbose) {
01194         Serial.print("flexpwm_wait "); Serial.print(timeout);
01195         Serial.print(" "); Serial.println(increment);
01196     }
01197
01198     while(flexpwm_running && !error_flag && elapsed < timeout) {
01199         delay(increment);

```

```

01200     elapsed += increment;
01201 }
01202 if (error_flag) {
01203     Serial.println("Error: flexpwm_wait error_flag is set.");
01204     return false;
01205 }
01206 if (elapsed >= timeout) {
01207     Serial.print("Error: flexpwm_wait timeout" );
01208     Serial.print(" elapsed "); Serial.print(elapsed);
01209     Serial.print(" timeout "); Serial.print(timeout);
01210     Serial.println(" millisecs");
01211     return false;
01212 }
01213 if (flexpwm_running) {
01214     Serial.print("Error: flexpwm_wait oops! Return with frameset still running");
01215     Serial.print(" elapsed "); Serial.print(elapsed);
01216     Serial.print(" timeout "); Serial.print(timeout);
01217     Serial.println(" millisecs");
01218     return false;
01219 }
01220
01221 if (verbose) {
01222     Serial.print("Success: flexpwm_wait");
01223     Serial.print(" running "); Serial.print(flexpwm_running);
01224     Serial.print(" error "); Serial.print(error_flag);
01225     Serial.print(" elapsed "); Serial.print(elapsed);
01226     Serial.print(" timeout "); Serial.print(timeout);
01227     Serial.println(" millisecs");
01228 }
01229
01230 return true;
01231 }
01232
01233 inline static bool pulse_armed = false;
01234
01235 static void pulse_sh_isr()
01236 {
01237     uint16_t status;
01238
01239     bool do_sync_toggle_here = false;
01240
01241     // this is a trailing edge interrupt := exposure timer
01242     uint64_t cyccnt64_now = cycles64();
01243
01244     // clear the interrupt
01245     status = flexpwm->SM[SH_SUBMODULE].STS;
01246     flexpwm->SM[SH_SUBMODULE].STS = status;
01247
01248 #if 1
01249     // =====
01250     // no clearing pulses
01251     if (!sh_clearing_counts) {
01252         // no more sh interrupts
01253         flexpwm->SM[SH_SUBMODULE].INTEN = 0;
01254
01255         // never on again until next start
01256         flexpwm->MCTRL |= FLEXPWM_MCTRL_CLDOK(SH_MASK);
01257         flexpwm->SM[SH_SUBMODULE].VAL2 = 0xFFFF;
01258         flexpwm->MCTRL |= FLEXPWM_MCTRL_LDOK(SH_MASK);
01259
01260         do_sync_toggle_here = true;
01261     }
01262
01263     // first pulse
01264     else if (!sh_clearing_counter) {
01265         /* shorten the period, this will happen
01266         after completing the present period. */
01267         flexpwm->MCTRL |= FLEXPWM_MCTRL_CLDOK(SH_MASK);
01268         flexpwm->SM[SH_SUBMODULE].VAL1 = sh_short_period_counts;
01269         flexpwm->MCTRL |= FLEXPWM_MCTRL_LDOK(SH_MASK);
01270
01271         // this is the end of the previous exposure
01272         do_sync_toggle_here = true;
01273
01274         // need to count
01275         sh_clearing_counter++;
01276     }
01277
01278 // last pulse?

```

```

01284     else if (++sh_clearing_counter == sh_clearing_counts) {
01285         // no more sh interrupts
01286         flexpwm->SM[SH_SUBMODULE].INTEN = 0;
01287
01288         // never on again until next start
01289         flexpwm->MCTRL |= FLEXPWM_MCTRL_CLDOK(SH_MASK);
01290         flexpwm->SM[SH_SUBMODULE].VAL2 = 0xFFFF;
01291         flexpwm->MCTRL |= FLEXPWM_MCTRL_LDOK(SH_MASK);
01292
01293         sh_clearing_counter = 0;
01294
01295         // this is the start of the next exposure
01296         do_sync_toggle_here = true;
01297     }
01298
01299     // for elapsed time and exposure
01300     sh_cyccnt64_prev = sh_cyccnt64_now;
01301     sh_cyccnt64_now = cyccnt64_now;
01302
01303     // sync is toggled on start and end of exposure
01304     if (sync_enabled && do_sync_toggle_here) {
01305         digitalToggleFast(SYNC_PIN);
01306         synctoggled = !synctoggled;
01307     }
01308
01309     // =====
01310 #else
01311     sh_clearing_counter++;
01312
01313     // Either no clearing pulses, or clearing is done
01314     if (sh_clearing_counter >= sh_clearing_counts) {
01315         // no more sh interrupts
01316         flexpwm->SM[SH_SUBMODULE].INTEN = 0;
01317
01318         // never on again until next start
01319         flexpwm->MCTRL |= FLEXPWM_MCTRL_CLDOK(SH_MASK);
01320         flexpwm->SM[SH_SUBMODULE].VAL2 = 0xFFFF;
01321         flexpwm->MCTRL |= FLEXPWM_MCTRL_LDOK(SH_MASK);
01322
01323         sh_clearing_counter = 0;
01324     }
01325
01326     // First clearing pulse
01327     else if (sh_clearing_counter == 1) {
01328         /* shorten the period, this will happen
01329         after completing the present period. */
01330         flexpwm->MCTRL |= FLEXPWM_MCTRL_CLDOK(SH_MASK);
01331         flexpwm->SM[SH_SUBMODULE].VAL1 = sh_short_period_counts;
01332         flexpwm->MCTRL |= FLEXPWM_MCTRL_LDOK(SH_MASK);
01333     }
01334
01335     // for elapsed time and exposure
01336     sh_cyccnt64_prev = sh_cyccnt64_now;
01337     sh_cyccnt64_now = cyccnt64_now;
01338
01339     // sync toggles on trailing edge of SH
01340     if (sync_enabled) {
01341         digitalToggleFast(SYNC_PIN);
01342         synctoggled = !synctoggled;
01343     }
01344 #endif
01345
01346     // bookkeeping
01347     sh_counter++;
01348
01349     // diagnostics
01350     if (!(status & CMPF_MASKA_OFF)) {
01351         oops_flag = true;
01352         Serial.println("OOPS! pulse sh without off bit set");
01353     }
01354 }
01355
01356 static void pulse_icg_isr()
01357 {
01358     uint16_t status;
01359
01360     status = flexpwm->SM[ICG_SUBMODULE].STS;
01361     flexpwm->SM[ICG_SUBMODULE].STS = status;
01362
01363     //Serial.print("icg_isr "); print_counters();
01364

```

```

01365 // -----
01366 // Start the cnvst clock
01367 flexpwm->MCTRL |= FLEXPWM_MCTRL_RUN(CNVST_MASK);
01368 // -----
01369
01370 // no more icg interrupts
01371 flexpwm->SM[ICG_SUBMODULE].INTEN = 0;
01372
01373 // never on again
01374 flexpwm->MCTRL |= FLEXPWM_MCTRL_CLDOK(ICG_MASK);
01375 flexpwm->SM[ICG_SUBMODULE].VAL2 = 0xFFFF;
01376 flexpwm->MCTRL |= FLEXPWM_MCTRL_LDOK(ICG_MASK);
01377
01378 flexpwm->SM[CLK_SUBMODULE].CTRL2 = FLEXPWM_SMCTRL2_FORCE; // force out while ldok, initializes the
counters
01379
01380 // here is our exposure, diff to most recent previous sh gate
01381 sh_cycCnt64_exposure = sh_cycCnt64_now - sh_cycCnt64_prev;
01382
01383 // bookkeeping
01384 icg_counter++;
01385
01386 // diagnostics
01387 if (!(status & CMPF_MASKA_OFF)) {
01388     oops_flag = true;
01389     Serial.println("OOPS! pulse icg without off bit set");
01390 }
01391 }
01392
01393 static void pulse_cnvst_isr()
01394 {
01395     uint16_t status;
01396
01397     status = flexpwm->SM[CNVST_SUBMODULE].STS;
01398     flexpwm->SM[CNVST_SUBMODULE].STS = status;
01399
01400     if ((status & CMPF_MASKA_OFF)) {
01401
01402         if (read_counter < read_counts) {
01403
01404             #ifdef ALLINONEBOARD
01405                 adc->adc0->startReadFast(ANALOGPIN);
01406                 while ( adc->adc0->isConverting() );
01407                 *read_pointer = adc->adc0->readSingle();
01408             #else
01409                 // Assert the convert pin
01410                 SETCNVST;
01411                 delayNanoseconds( 670 ); // 710 nanoseconds minus spi setup time
01412                 CLEARCNVST; // need 30 nanoseconds after this
01413
01414                 *read_pointer = SPI.transfer16(0xFFFF);
01415                 *read_pointer ^= (0x1<15);
01416             #endif
01417
01418             // bookkeeping for the read
01419             read_pointer++;
01420             read_counter++;
01421
01422             // The read is done
01423             if (read_counter == read_counts) {
01424
01425                 // no more cnvst interrupts
01426                 flexpwm->SM[CNVST_SUBMODULE].INTEN = 0;
01427
01428                 // Stop everything on this flexpwm
01429                 // ** Indicate stop only after callback **
01430                 flexpwm->MCTRL |= FLEXPWM_MCTRL_CLDOK(0xF); // clear ldok for all submodules
01431                 flexpwm->MCTRL = 0; // clear run for all submodules
01432
01433                 // -----
01434                 // User supplied function, per read complete
01435                 // Recommend that the user ignore frame 0
01436                 if (read_callback) {
01437                     read_callback();
01438                 }
01439
01440                 // ** Now we can indicate stopped **
01441                 flexpwm_running = false;
01442
01443                 // =====
01444                 // Frame bookkeeping, re-arm etc

```

```
01445         frame_counter++;
01446
01447         // More frames?
01448         if (frame_counter < frame_counts) {
01449             pulse_arm();
01450         }
01451
01452         // Nope, we're done with this frameset
01453         else if (frame_counter == frame_counts) {
01454
01455             if (frames_completed_callback) {
01456                 frames_completed_callback();
01457             }
01458
01459             // frameset completed, update frameset counter
01460             frameset_counter++;
01461
01462             // More framesets?
01463             if (frameset_counter < frameset_counts) {
01464                 pulse_init_frames(); // init the icg counter too.
01465             }
01466
01467             // Nope, we're done with all of the framesets
01468             else if (frameset_counter == frameset_counts) {
01469
01470                 if (framesets_completed_callback) {
01471                     framesets_completed_callback();
01472                 }
01473
01474                 // reset sync pin
01475                 if (synctoggled) {
01476                     digitalToggleFast(SYNC_PIN);
01477                     synctoggled = false;
01478                 }
01479
01480                 // and get ready for the next frame set
01481                 pulse_init_frameset();
01482             }
01483
01484             // oops, shouldn't get here
01485             else {
01486                 oops_flag = true;
01487                 Serial.println("Oops! cnvst interrupt after frameset_counts complete");
01488             }
01489
01490         }
01491
01492         // oops, shouldn't get here
01493         else {
01494             oops_flag = true;
01495             Serial.println("Oops! cnvst interrupt after frame_counts complete");
01496         }
01497     }
01498 }
01499
01500 }
01501
01502 // oops, shouldn't get here
01503 else {
01504     oops_flag = true;
01505     Serial.println("OOPS! pulse cnvst interrupt after read_counts complete");
01506 }
01507
01508 }
01509
01510 // oops, shouldn't get here
01511 else {
01512     oops_flag = true;
01513     Serial.print("OOPS! pulse cnvst without off bit set ");
01514     Serial.println(status, HEX);
01515 }
01516
01517 // diagnostics
01518 cnvst_counter++;
01519
01520 }
01521
01522 static void pulse_start()
01523 {
01524     if (pulse_armed) {
01525         sh_clearing_counter = 0;
```

```

01526     flexpwm_start();
01527 }
01528 else {
01529     stop_runs_only();
01530     Serial.println("Error: pulse_start() - but not armed");
01531     error_flag = true;
01532 }
01533 }
01534
01535 static void pulse_arm()
01536 {
01537     uint16_t status;
01538
01539     if (!busytoggled) {
01540         busytoggled = true;
01541         digitalToggleFast(BUSY_PIN);
01542     }
01543
01544     read_pointer = read_buffer;
01545     read_counter = 0;
01546
01547     sh_clearing_counter = 0;
01548
01549     // Restore the SH submodule counter and interrupt enable
01550     status = flexpwm->SM[SH_SUBMODULE].STS;
01551     flexpwm->SM[SH_SUBMODULE].STS = status;
01552
01553     flexpwm->SM[SH_SUBMODULE].INTEN = CMPF_MASKA_OFF;
01554
01555     flexpwm->MCTRL |= FLEXPWM_MCTRL_CLDOK(SH_MASK);
01556     flexpwm->SM[SH_SUBMODULE].VAL1 = sh.period_counts - 1;
01557     flexpwm->SM[SH_SUBMODULE].VAL2 = sh.onA_counts;
01558     flexpwm->SM[SH_SUBMODULE].VAL3 = sh.offA_counts;
01559     flexpwm->MCTRL |= FLEXPWM_MCTRL_LDOK(SH_MASK);
01560
01561     // Restore the ICG submodule counter and interrupt enable
01562     status = flexpwm->SM[ICG_SUBMODULE].STS;
01563     flexpwm->SM[ICG_SUBMODULE].STS = status;
01564
01565     flexpwm->SM[ICG_SUBMODULE].INTEN = CMPF_MASKA_OFF;
01566
01567     flexpwm->MCTRL |= FLEXPWM_MCTRL_CLDOK(ICG_MASK);
01568     flexpwm->SM[ICG_SUBMODULE].VAL2 = icg.onA_counts;
01569     flexpwm->SM[ICG_SUBMODULE].VAL3 = icg.offA_counts;
01570     flexpwm->MCTRL |= FLEXPWM_MCTRL_LDOK(ICG_MASK);
01571
01572     // Restore the CNVST submodule interrupt enable
01573     status = flexpwm->SM[CNVST_SUBMODULE].STS;
01574     flexpwm->SM[CNVST_SUBMODULE].STS = status;
01575
01576     flexpwm->SM[CNVST_SUBMODULE].INTEN = CMPF_MASKA_OFF;
01577
01578 #ifdef DEBUG
01579     Serial.println("pulse armed");
01580 #endif
01581     pulse_armed = true;
01582 }
01583
01584 static void pulse_init_frames()
01585 {
01586     sh_cyccnt64_start = 0;
01587     sh_cyccnt64_prev = 0;
01588     sh_cyccnt64_now = 0;
01589
01590     sh_counter = 0;
01591     icg_counter = 0;
01592     cnvst_counter = 0;
01593
01594     sh_clearing_counter = 0;
01595
01596     frame_counter = 0;
01597
01598     pulse_arm();
01599
01600 #ifdef DEBUG
01601     Serial.println("pulse init frames");
01602 #endif
01603 }
01604
01605 static void pulse_init_frameset()
01606 {

```



```

01607     pulse_init_frames();
01608
01609     frameset_counter = 0;
01610
01611     pulse_arm();
01612 }
01613
01614 bool setup_pulse(float clk_secs, float sh_secs, float sh_offset_secs, float icg_secs, float
icg_offset_secs,
01615                 uint16_t *buffer, unsigned int nbuffer, void (*callbackf)())
01616 {
01617     char cbuffer[128] = {0};
01618     unsigned int cnvst_delay = 1;
01619
01620     // stop everything and disconnect irqs
01621     stop_with_irqs();
01622
01623     // clear the error flags
01624     clear_error_flags();
01625
01626     // clear run and trigger mode
01627     clear_mode();
01628
01629     // clear sync and busy pins
01630     clear_sync_busy_pins();
01631
01632
01633     sprintf(cbuffer,
01634            "tcd1304 setup pulse with clk %.5gs sh %.5gs offset %.5gs icg offset %.5gs",
01635            clk_secs, sh_secs, sh_offset_secs, icg_offset_secs);
01636     Serial.println(cbuffer);
01637
01638     // clear elapsed times
01639     sh_cyccnt64_start = 0;
01640     sh_cyccnt64_prev = 0;
01641     sh_cyccnt64_now = 0;
01642
01643     timer_cyccnt64_start = 0;
01644     timer_cyccnt64_prev = 0;
01645     timer_cyccnt64_now = 0;
01646
01647     trigger_cyccnt64_start = 0;
01648     trigger_cyccnt64_prev = 0;
01649     trigger_cyccnt64_now = 0;
01650
01651     // clear bookkeeping counters
01652     sh_counter = 0;
01653     icg_counter = 0;
01654     cnvst_counter = 0;
01655
01656     sh_clearing_counter = 0;
01657
01658     // default to 1 frame and 1 frameset
01659     frame_counter = 0;
01660     frame_counts = 1;
01661
01662     frameset_counter = 0;
01663     frameset_counts = 1;
01664
01665     // -----
01666     // Setup the master clock
01667
01668     if (clk_secs < 0.25E-6 || clk_secs > 1.25E-6) {
01669         sprintf(cbuffer,
01670                "Error: setup_flexpwm_single_sequence with clk interval %.6g secs out of range 1E-6 to
5.E-6",
01671                clk_secs);
01672         Serial.println(cbuffer);
01673         return false;
01674     }
01675
01676     clk.prescale = 0;
01677     clk.divider = (1<<clk.prescale);
01678
01679     clk.period_counts = ceil(clk_secs*F_BUS_ACTUAL/clk.divider);
01680
01681     clk.onA_counts = 0;
01682     clk.offA_counts = (clk.period_counts/2);
01683     clk.invertA = false;
01684
01685     clk.onB_counts = 0;

```

```

01686     clk.offB_counts    = 0;
01687     clk.invertB        = false;
01688
01689     clk.inten_mask     = 0;
01690
01691     clk.ctrl2_mask     = PWM_CTRL2_CLOCK_MASTER;
01692
01693     // -----
01694     // Setup the CNVST clock, 4 times the clock period,
01695     cnvst.prescale = clk.prescale;
01696     cnvst.divider = (1<cnvst.prescale);
01697
01698     cnvst.period_counts = 4*clk.period_counts;
01699
01700     cnvst.onA_counts    = (cnvst_delay + cnvst_extra_delay_counts)*clk.period_counts;    // allow extra
delay for testing
01701     cnvst.offA_counts   = cnvst.onA_counts + ceil(CNVST_PULSE_SECS*F_BUS_ACTUAL/cnvst.divider);
01702     cnvst.invertA       = false;
01703
01704     cnvst.onB_counts    = 0;
01705     cnvst.offB_counts   = 0;
01706     cnvst.invertB       = false;
01707
01708     //cnvst.ctrl2_mask   = PWM_CTRL2_CLOCK_SYNC;
01709     cnvst.ctrl2_mask    = PWM_CTRL2_CLOCK_MASTER; // needs to be master, because run bit is disabled when
running from clk's clock.
01710
01711     // -----
01712     // setup the sh, sh_offset-icg_offset is the icg-sh timing, runs once, isr has to set on,off to 0xffff
01713     sh.prescale = clk.prescale;
01714     sh.divider = (1<sh.prescale);
01715
01716     sh.period_counts    = 0x8000;
01717     sh.onA_counts       = ceil(sh_offset_secs*F_BUS_ACTUAL/sh.divider);
01718     sh.offA_counts      = sh.onA_counts + ceil(sh_secs*F_BUS_ACTUAL/sh.divider);
01719     sh.invertA          = false;
01720
01721     sh.onB_counts       = 0;
01722     sh.offB_counts      = 0;
01723     sh.invertB          = false;
01724
01725     sh.ctrl2_mask       = PWM_CTRL2_CLOCK_SLAVE;
01726     //sh.ctrl2_mask     = FLEXPWM_SMCTRL2_CLK_SEL(0x2);
01727
01728     // -----
01729     // setup icg, icg interrupt has to stop sh and icg by moving val2 to > val1
01730     icg.prescale = clk.prescale;
01731     icg.divider = (1<icg.prescale);
01732
01733     icg.period_counts   = 0x8000;
01734     icg.onA_counts      = ceil(icg_offset_secs*F_BUS_ACTUAL/icg.divider);
01735     icg.offA_counts     = icg.onA_counts + ceil(icg_secs*F_BUS_ACTUAL/icg.divider);
01736     icg.invertA         = true;
01737
01738     icg.onB_counts      = 0;
01739     icg.offB_counts     = 0;
01740     icg.invertB         = false;
01741
01742     icg.ctrl2_mask      = PWM_CTRL2_CLOCK_SLAVE;
01743     //icg.ctrl2_mask    = FLEXPWM_SMCTRL2_CLK_SEL(0x2);
01744
01745     // if we are going to run some clearing cycles on the shift gate
01746     if (sh_clearing_counts) {
01747         // make sure it comes after the icg isr and before the first cnvst isr
01748         sh.period_counts = icg.offA_counts + clk.period_counts;
01749     }
01750
01751     // if we go to short clearing cycles, this is the period
01752     sh_short_period_counts = 2*(sh.offA_counts - sh.onA_counts) + 1;
01753     if (sh_short_period_counts <= sh.offA_counts) {
01754         sh_short_period_counts = sh.offA_counts + 1;
01755     }
01756
01757     // -----
01758     Serial.println("");
01759     if (!print_and_check_submodule(&clk)) {
01760         return false;
01761     }
01762
01763     Serial.println("");
01764     if (!print_and_check_submodule(&icg)) {

```

```

01765         return false;
01766     }
01767
01768     Serial.println("");
01769     if (!print_and_check_submodule(&sh)) {
01770         return false;
01771     }
01772
01773     Serial.println("");
01774     if (!print_and_check_submodule(&cnvst)) {
01775         return false;
01776     }
01777
01778     Serial.println("");
01779     Serial.print("flexpwm: clearing_pulses ");
01780     Serial.println(sh_clearing_counts);
01781
01782     // -----
01783     load_submodule(&clk);
01784     load_submodule(&icg);
01785     load_submodule(&sh);
01786     load_submodule(&cnvst);
01787
01788     // -----
01789     // Setup the readout buffer
01790     read_buffer = buffer;
01791     read_counts = nbuffer;
01792
01793     read_pointer = read_buffer;
01794     read_counter = 0;
01795
01796     read_callback = callbackf;
01797
01798     // -----
01799     attach_isr( &sh, CMPF_MASKA_OFF, pulse_sh_isr);
01800     attach_isr( &icg, CMPF_MASKA_OFF, pulse_icg_isr);
01801     attach_isr( &cnvst, CMPF_MASKA_OFF, pulse_cnvst_isr);
01802
01803     // -----
01804     // stuff that other routines need to know
01805     mode = PULSE;
01806     pulse_armed = true;
01807     trigger_mode = false;
01808
01809     // timer needs this
01810     timer_interframe_min_secs = (icg.offA_counts/F_BUS_ACTUAL)*icg.divider;    // offset before read
01811 starts timer_interframe_min_secs += ((float) cnvst.period_counts/F_BUS_ACTUAL)*cnvst.divider * read_counts;
01812 // read time timer_interframe_min_secs += USBTRANSFERSECS;
01813
01814     // round up to 1 msec
01815     timer_interframe_min_secs = ceil((timer_interframe_min_secs+1.E-3)/1.E-3)*1.E-3;
01816
01817     Serial.print("TCD1304 setup_pulse success, min frame interval ");
01818     Serial.print(timer_interframe_min_secs,6);
01819     Serial.println(" secs");
01820
01821     return true;
01822 }
01823
01824 // Frameset, short exposure times
01825
01826 inline static bool frameset_armed = false;
01827
01828 static void frameset_sh_isr()
01829 {
01830     uint16_t status;
01831     unsigned int remainder;
01832
01833     // this is a trailing edge interrupt := exposure timer
01834     uint64_t cyccnt64_now = cycles64();
01835
01836     // clear the interrupt
01837     status = flexpwm->SM[SH_SUBMODULE].STS;
01838     flexpwm->SM[SH_SUBMODULE].STS = status;
01839
01840     // this is where we are relative to the icg pulse
01841     remainder = sh_counter % sh_counts_per_icg;
01842
01843     // one test for both pulses beginning and ending the sampling interval

```

```

01846     if (remainder <= 1) {
01847         if (sync_enabled) {
01848             digitalToggleFast(SYNC_PIN);
01849             synctoggled = !synctoggled;
01850         }
01851         sh_cyccnt64_prev = sh_cyccnt64_now;
01852         sh_cyccnt64_now = cyccnt64_now;
01853         sh_cyccnt64_exposure = cyccnt64_now - sh_cyccnt64_prev;
01854     }
01855     // bookkeeping
01856     sh_counter++;
01857
01858     // diagnostics
01859     if (!(status & CMPF_MASKA_OFF)) {
01860         oops_flag = true;
01861         Serial.println("OOPS! pulse sh without off bit set");
01862     }
01863 }
01864
01865 static void frameset_icg_isr()
01866 {
01867     uint16_t status;
01868
01869     status = flexpwm->SM[ICG_SUBMODULE].STS;
01870     flexpwm->SM[ICG_SUBMODULE].STS = status;
01871
01872     //Serial.print("icg_isr "); print_counters();
01873
01874     // -----
01875     // Start the cnvst clock
01876     flexpwm->MCTRL |= FLEXPWM_MCTRL_RUN(CNVST_MASK);
01877     // -----
01878
01879     // bookkeeping
01880     icg_counter++;
01881
01882     // diagnostics
01883     if (!(status & CMPF_MASKA_OFF)) {
01884         oops_flag = true;
01885         Serial.println("OOPS! pulse icg without off bit set");
01886     }
01887 }
01888
01889 static void frameset_cnvst_isr()
01890 {
01891     uint16_t status;
01892
01893     status = flexpwm->SM[CNVST_SUBMODULE].STS;
01894     flexpwm->SM[CNVST_SUBMODULE].STS = status;
01895
01896     if ((status & CMPF_MASKA_OFF)) {
01897         if (read_counter < read_counts) {
01898
01899 #ifdef ALLINONEBOARD
01900             adc->adc0->startReadFast(ANALOGPIN);
01901             while ( adc->adc0->isConverting() );
01902             *read_pointer = adc->adc0->readSingle();
01903 #else
01904             // Assert the convert pin
01905             SETCNVST;
01906             delayNanoseconds( 670 ); // 710 nanoseconds minus spi setup time
01907             CLEARCNVST; // need 30 nanoseconds after this
01908
01909             *read_pointer = SPI.transfer16(0xFFFF);
01910             *read_pointer ^= (0x1<15);
01911 #endif
01912
01913             // bookkeeping for the read
01914             read_pointer++;
01915             read_counter++;
01916
01917             // The read is done
01918             if (read_counter == read_counts) {
01919                 // stop cnvst
01920                 flexpwm->MCTRL &= ~FLEXPWM_MCTRL_RUN(CNVST_MASK);
01921
01922                 // -----
01923                 // User supplied function, per read complete

```

```
01927         // Recommend that the user ignore frame 0
01928         if (read_callback) {
01929             read_callback();
01930         }
01931
01932         // =====
01933         // Frame bookkeeping
01934         frame_counter++;
01935
01936         // More frames?
01937         if (frame_counter < frame_counts) {
01938             frameset_arm();
01939         }
01940
01941         // Are we done with this frameset?
01942         else if (frame_counter == frame_counts) {
01943
01944             // stop everything on this flexpwm
01945             flexpwm_stop();
01946
01947             if (frames_completed_callback) {
01948                 frames_completed_callback();
01949             }
01950
01951             // frameset completed, update frameset counter
01952             frameset_counter++;
01953
01954             // More framesets?
01955             if (frameset_counter < frameset_counts) {
01956                 frameset_init_frames(); // init the icg counter too.
01957             }
01958
01959             // Nope, we're done with all of the framesets
01960             else if (frameset_counter == frameset_counts) {
01961
01962                 if (framesets_completed_callback) {
01963                     framesets_completed_callback();
01964                 }
01965
01966                 // reset sync pin
01967                 if (synctoggled) {
01968                     digitalToggleFast(SYNC_PIN);
01969                     synctoggled = false;
01970                 }
01971
01972                 // and get ready for the next frame set
01973                 frameset_init_frameset();
01974
01975             }
01976
01977             // oops, shouldn't get here
01978             else {
01979                 oops_flag = true;
01980                 Serial.println("Oops!  cnvst interrupt after frameset_counts complete");
01981             }
01982
01983         }
01984
01985         // oops, shouldn't get here
01986         else if (frame_counter > frame_counts) {
01987             oops_flag = true;
01988             Serial.println("Oops!  cnvst interrupt after frame_counts complete.");
01989         }
01990     }
01991 }
01992
01993 // oops, shouldn't get here
01994 else {
01995     oops_flag = true;
01996     Serial.println("OOPS!  pulse cnvst interrupt after read_counts complete");
01997 }
01998
01999 }
02000
02001 // oops, shouldn't get here
02002 else {
02003     oops_flag = true;
02004     Serial.print("OOPS!  pulse cnvst without off bit set ");
02005     Serial.println(status, HEX);
02006 }
02007
```

```

02008     // diagnostics
02009     cnvst_counter++;
02010
02011 }
02012
02013
02014 static void frameset_start()
02015 {
02016     if (flexpwm_running) {
02017         error_flag = true;
02018         Serial.println("Error: framset already running");
02019         return;
02020     }
02021
02022     if (!frameset_armed) {
02023         error_flag = true;
02024         Serial.println("Error: framset not armed");
02025         return;
02026     }
02027
02028     timer_cyccnt64_start = cycles64();
02029     timer_cyccnt64_prev = timer_cyccnt64_start;
02030     timer_cyccnt64_now = timer_cyccnt64_start;
02031
02032     // start the clk, sh, icg
02033     flexpwm_start();
02034 }
02035
02036
02037 // -----
02038 static void frameset_arm()
02039 {
02040     if (!busy toggled) {
02041         busy toggled = true;
02042         digitalWriteFast(BUSY_PIN);
02043     }
02044
02045     read_pointer = read_buffer;
02046     read_counter = 0;
02047
02048     frameset_armed = true;
02049 }
02050
02051 static void frameset_init_frames()
02052 {
02053     sh_cyccnt64_start = 0;
02054     sh_cyccnt64_prev = 0;
02055     sh_cyccnt64_now = 0;
02056
02057     sh_counter = 0;
02058     icg_counter = 0;
02059     cnvst_counter = 0;
02060
02061     frame_counter = 0;
02062
02063     sh_clearing_counter = 0;
02064
02065     frameset_arm();
02066 }
02067
02068 static void frameset_init_frameset()
02069 {
02070     frameset_init_frames();
02071
02072     frameset_counter = 0;
02073
02074     frameset_arm();
02075 }
02076
02077 bool setup_frameset(float clk_secs, float sh_secs, float sh_offset_secs, float icg_secs, float
02078 icg_offset_secs,
02079                     float exposure_secs, float frame_interval_secs, unsigned int nframes,
02080                     uint16_t *buffer, unsigned int nbuffer, void (*callbackf)())
02081 {
02082     //char cbuffer[128] = {0};
02083     unsigned int cnvst_delay = 1;
02084
02085     unsigned int prescale = 0;
02086     unsigned int divider = 1;
02087

```

```

02088     float clock_period_secs;
02089
02090     unsigned int clock_period_counts;
02091     unsigned int exposure_period_counts;
02092     unsigned int frame_period_counts;
02093
02094
02095     // -----
02096     // stop everything and disable irqs
02097     stop_with_irqs();
02098
02099     // clear sync and busy pins
02100     clear_sync_busy_pins();
02101
02102     // clear the error flags
02103     clear_error_flags();
02104
02105     // clear run mode and trigger mode
02106     clear_mode();
02107
02108     // -----
02109     // announce ourselves
02110     Serial.print("#tcd1304 setup frameset, clk "); Serial.print(clk_secs,8);
02111     Serial.print(" sh "); Serial.print(sh_secs,8);
02112     Serial.print(" offset "); Serial.print(sh_offset_secs,8);
02113     Serial.print(" icg "); Serial.print(icg_secs,8);
02114     Serial.print(" offset "); Serial.print(icg_offset_secs,8);
02115     Serial.print(" exposure "); Serial.print(exposure_secs,6);
02116     Serial.print(" interval "); Serial.print(frame_interval_secs,6);
02117     Serial.print(" frames "); Serial.println(nframes);
02118
02119     // -----
02120     // clear elapsed times
02121     sh_cyccnt64_start = 0;
02122     sh_cyccnt64_prev = 0;
02123     sh_cyccnt64_now = 0;
02124
02125     timer_cyccnt64_start = 0;
02126     timer_cyccnt64_prev = 0;
02127     timer_cyccnt64_now = 0;
02128
02129     trigger_cyccnt64_start = 0;
02130     trigger_cyccnt64_prev = 0;
02131     trigger_cyccnt64_now = 0;
02132
02133     // clear bookkeeping counters
02134     sh_counter = 0;
02135     icg_counter = 0;
02136     cnvst_counter = 0;
02137
02138     sh_clearing_counter = 0;
02139
02140     // default to 1 frame and 1 frameset
02141     frame_counter = 0;
02142     //frame_counts = nframes ? nframes : 10;
02143     frame_counts = nframes ? nframes+1 : 10;
02144
02145     frameset_counter = 0;
02146     frameset_counts = 1;
02147
02148     // -----
02149     // Some basic checks
02150     if (exposure_secs*2 > frame_interval_secs) {
02151         Serial.println("Error: requested exposure > requested frame interval/2");
02152         error_flag = true;
02153         return false;
02154     }
02155
02156     if (clk_secs > exposure_secs) {
02157         Serial.println("Error: requested clk > requested exposure");
02158         error_flag = true;
02159         return false;
02160     }
02161
02162     if (clk_secs < 0.25E-6) {
02163         Serial.println("Error: requested clk < 0.25E-6 (4MHz)");
02164         error_flag = true;
02165         return false;
02166     }
02167
02168     if (clk_secs > 1.25E-6) {

```

```

02169     Serial.println("Error: requested clk > 1.25E-6 (800kHz)");
02170     error_flag = true;
02171     return false;
02172 }
02173
02174 // -----
02175 // need to find divider for the frame interval
02176 while (frame_interval_secs * (F_BUS_ACTUAL/divider) > 65533) { // reserve two time slots, 0xFFFF - 2
02177     if (prescale >= 7) {
02178         Serial.println("Error: requested frame interval is too large");
02179         error_flag = true;
02180         return false;
02181     }
02182     prescale++;
02183     divider = (1<<prescale);
02184 }
02185 Serial.print("#Exposure prescale "); Serial.print(prescale);
02186 Serial.print(" divider "); Serial.println(divider);
02187
02188 clock_period_counts = ceil(clk_secs*F_BUS_ACTUAL);
02189 clock_period_counts = ceil(clock_period_counts/divider) * divider;
02190 clock_period_secs = (float)clock_period_counts/F_BUS_ACTUAL;
02191 Serial.print("#Clock period counts "); Serial.print(clock_period_counts);
02192 Serial.print(" secs "); Serial.println(clock_period_secs,8);
02193
02194 if ((clock_period_counts < 4) || (clock_period_secs > (1./TCD1304_MINCLKHZ))) || (clock_period_secs <
02195 (1./TCD1304_MAXCLKHZ))) {
02196     Serial.println("Error: not able to support this combination of clock and exposure times.");
02197     error_flag = true;
02198     return false;
02199 }
02199 Serial.print("#Clock period counts "); Serial.println(clock_period_counts);
02200
02201 exposure_period_counts = ceil(exposure_secs*F_BUS_ACTUAL);
02202 exposure_period_counts = (exposure_period_counts/clock_period_counts) * clock_period_counts; //
already a multiple of divider
02203 Serial.print("#Exposure period counts "); Serial.println(exposure_period_counts);
02204
02205 frame_period_counts = ceil(frame_interval_secs*F_BUS_ACTUAL);
02206 frame_period_counts = (frame_period_counts/exposure_period_counts) * exposure_period_counts; //
already a multiple of divider
02207 Serial.print("#Frame period counts "); Serial.println(frame_period_counts);
02208
02209 sh_counts_per_icg = frame_period_counts/exposure_period_counts;
02210 Serial.print("#Exposure to frame ratio ");
02211 Serial.println((float)frame_period_counts/exposure_period_counts);
02212
02213 // -----
02214 // Setup the master clock, always with divider = 1
02215 clk.prescale = 0;
02216 clk.divider = 1;
02217
02218 clk.period_counts = clock_period_counts;
02219
02220 clk.onA_counts = 0;
02221 clk.offA_counts = (clk.period_counts/2);
02222 clk.invertA = false;
02223
02224 clk.onB_counts = 0;
02225 clk.offB_counts = 0;
02226 clk.invertB = false;
02227
02228 clk.inten_mask = 0;
02229
02230 clk.ctrl2_mask = PWM_CTRL2_CLOCK_MASTER;
02231
02232 // -----
02233 // Setup the CNVST clock, 4 times the clock period, on the same clock divider
02234 cnvst.prescale = clk.prescale;
02235 cnvst.divider = clk.divider;
02236
02237 cnvst.period_counts = 4*clk.period_counts;
02238
02239 cnvst.onA_counts = (cnvst_delay + cnvst_extra_delay_counts)*clk.period_counts; // allow extra
delay for testing
02240 cnvst.offA_counts = cnvst.onA_counts + ceil(CNVST_PULSE_SECS*F_BUS_ACTUAL/cnvst.divider);
02241 cnvst.invertA = false;
02242
02243 cnvst.onB_counts = 0;
02244 cnvst.offB_counts = 0;

```



```

02245     cnvst.invertB      = false;
02246
02247     //cnvst.ctrl2_mask  = PWM_CTRL2_CLOCK_SYNC;
02248     cnvst.ctrl2_mask    = PWM_CTRL2_CLOCK_MASTER; // needs to be master, because run bit is disabled when
running from clk's clock.
02249
02250     // -----
02251     // setup the sh, sh_offset-icg_offset is the icg-sh timing, runs once, isr has to set on,off to 0xffff
02252     sh.prescale         = prescale;
02253     sh.divider          = divider;
02254
02255     sh.period_counts    = exposure_period_counts/divider;
02256
02257     sh.onA_counts       = ceil(sh_offset_secs*F_BUS_ACTUAL/sh.divider);
02258     sh.offA_counts      = sh.onA_counts + ceil(sh_secs*F_BUS_ACTUAL/sh.divider);
02259     sh.invertA          = false;
02260
02261     sh.onB_counts       = 0;
02262     sh.offB_counts      = 0;
02263     sh.invertB          = false;
02264
02265     sh.ctrl2_mask       = PWM_CTRL2_CLOCK_SLAVE;
02266     //sh.ctrl2_mask     = FLEXPWM_SMCTRL2_CLK_SEL(0x2);
02267
02268     // -----
02269     // setup icg, icg interrupt has to stop sh and icg by moving val2 to > val1
02270     icg.prescale        = prescale;
02271     icg.divider         = divider;
02272
02273     icg.period_counts   = frame_period_counts/divider;
02274
02275     // icg happens on the second sh pulse
02276     icg.onA_counts      = sh.period_counts + ceil(icg_offset_secs*F_BUS_ACTUAL/icg.divider);
02277     icg.offA_counts     = icg.onA_counts + ceil(icg_secs*F_BUS_ACTUAL/icg.divider);
02278     icg.invertA         = true;
02279
02280     icg.onB_counts      = 0;
02281     icg.offB_counts     = 0;
02282     icg.invertB         = false;
02283
02284     icg.ctrl2_mask      = PWM_CTRL2_CLOCK_SLAVE;
02285     //icg.ctrl2_mask    = FLEXPWM_SMCTRL2_CLK_SEL(0x2);
02286
02287     // -----
02288     Serial.println("");
02289     if (!print_and_check_submodule(&clk)) {
02290         error_flag = true;
02291         return false;
02292     }
02293
02294     Serial.println("");
02295     if (!print_and_check_submodule(&icg)) {
02296         error_flag = true;
02297         return false;
02298     }
02299
02300     Serial.println("");
02301     if (!print_and_check_submodule(&sh)) {
02302         error_flag = true;
02303         return false;
02304     }
02305
02306     Serial.println("");
02307     if (!print_and_check_submodule(&cnvst)) {
02308         error_flag = true;
02309         return false;
02310     }
02311
02312     // -----
02313     load_submodule(&clk);
02314     load_submodule(&icg);
02315     load_submodule(&sh);
02316     load_submodule(&cnvst);
02317
02318     // -----
02319     // Setup the readout buffer
02320     read_buffer         = buffer;
02321     read_counts         = nbuffer;
02322
02323     read_pointer        = read_buffer;
02324     read_counter        = 0;

```

```

02325
02326     read_callback = callbackf;
02327
02328     // -----
02329     attach_isr( &sh, CMPF_MASKA_OFF, frameset_sh_isr);
02330     attach_isr( &icg, CMPF_MASKA_OFF, frameset_icg_isr);
02331     attach_isr( &cnvst, CMPF_MASKA_OFF, frameset_cnvst_isr);
02332
02333     // -----
02334     // stuff that other routines need to know
02335     mode = FRAMESET;
02336     trigger_mode = false;
02337
02338     frameset_armed = true;
02339
02340     return true;
02341 }
02342
02343 // =====
02344 // #define DEBUG_TCD1304_TIMER
02345
02346 inline static bool timer_first_time_flag = true;
02347 inline static bool timer_running = false;
02348
02349 static void timer_isr()
02350 {
02351     volatile uint16_t status;
02352
02353     uint64_t cyccnt64_now = cycles64();
02354
02355     status = timerflexpwm->SM[TIMER_SUBMODULE].STS;
02356     timerflexpwm->SM[TIMER_SUBMODULE].STS = status;
02357
02358     // spurious first invocation
02359     if (!status) {
02360         if (timer_inner_counter) {
02361             error_flag = true;
02362             timer_stop();
02363             Serial.println("Error: OOPS! pulse timer with null status, stopping ");
02364         }
02365         return;
02366     }
02367
02368     timer_inner_counter++;
02369
02370     // completed the inner counter, call the callback
02371     if (timer_first_time_flag || timer_inner_counter == timer_inner_counts) {
02372
02373         timer_cyccnt64_prev = timer_cyccnt64_now;
02374         timer_cyccnt64_now = cyccnt64_now;
02375
02376         if (timer_callback) {
02377             timer_callback();
02378         }
02379
02380         // Reset the inner counter
02381         timer_inner_counter = 0;
02382
02383         // clear the first time flag
02384         timer_first_time_flag = false;
02385
02386         // Update and check the outer counter, are we done?
02387         timer_outer_counter++;
02388         if (timer_outer_counter == timer_outer_counts) {
02389             // stop the timer only, we might be interrupt driven
02390             timer_stop();
02391         }
02392     }
02393
02394     // diagnostics
02395     if (!(status & timer.intena_mask)) {
02396         timer_stop();
02397         error_flag = true;
02398         Serial.print("Error: OOPS! pulse timer without the right bit, status ");
02399         Serial.println(status, HEX);
02400         print_counters();
02401     }
02402 }
02403
02404 static void timer_start()
02405 {

```

```

02406     if (timer_running) {
02407         error_flag = true;
02408         Serial.println("Error: timer_start() already running");
02409         return;
02410     }
02411
02412     timer_cyccnt64_start = cycles64();
02413     timer_cyccnt64_prev = timer_cyccnt64_start;
02414     timer_cyccnt64_now = timer_cyccnt64_start;
02415
02416     if (pulse_armed && frame_counts && timer_inner_counts && timer_outer_counts) {
02417
02418         oops_flag = false;
02419         error_flag = false;
02420
02421         timer_running = true; // do it before the start, in case isr changes it
02422
02423         frame_counter = 0;
02424         timer_inner_counter = 0;
02425         timer_outer_counter = 0;
02426         timer_first_time_flag = true;
02427
02428         timerflexpwm->SM[TIMER_SUBMODULE].STS = 0xFF; // clear all interrupt bits
02429
02430         timerflexpwm->MCTRL |= TIMER_MASK; // set load ok for timer submodule
02431         timerflexpwm->MCTRL |= (TIMER_MASK << 8); // set run for timer submodule
02432     }
02433     else {
02434
02435         error_flag = true;
02436
02437         Serial.print("Error: timer_start() - but not setup ");
02438         Serial.print(" pulse armed "); Serial.print(pulse_armed);
02439         Serial.print(" frame_counts "); Serial.print(frame_counts);
02440         Serial.print(" timer_inner_counts "); Serial.print(timer_inner_counts);
02441         Serial.print(" timer_outer_counts "); Serial.println(timer_inner_counts);
02442
02443         timer_stop();
02444     }
02445 }
02446
02447 static void timer_stop()
02448 {
02449     timerflexpwm->MCTRL |= FLEXPWM_MCTRL_CLDOK(TIMER_MASK);
02450     timerflexpwm->MCTRL = 0;
02451     timer_running = false;
02452 }
02453
02454 static void timer_stop_with_irq()
02455 {
02456     timerflexpwm->MCTRL |= FLEXPWM_MCTRL_CLDOK(TIMER_MASK);
02457     timerflexpwm->MCTRL = 0;
02458     timer_running = false;
02459     NVIC_DISABLE_IRQ(TIMER_IRQ);
02460 }
02461
02462 bool timer_wait(float timeout=1., float timestep=0.01, bool verbose=false)
02463 {
02464     uint32_t timeout = (uint) (timeout_ * 1000);
02465     uint32_t increment = (uint) (timestep_ * 1000);
02466     uint32_t elapsed = 0;
02467
02468     if (verbose) {
02469         Serial.print("timer_wait "); Serial.print(timeout);
02470         Serial.print(" "); Serial.println(increment);
02471     }
02472
02473     while(timer_running && !error_flag && elapsed < timeout) {
02474         delay(increment);
02475         elapsed += increment;
02476     }
02477     if (error_flag) {
02478         Serial.println("Error: timer_wait error_flag is set.");
02479         return false;
02480     }
02481     if (elapsed >= timeout) {
02482         Serial.print("Error: timer_wait timeout ");
02483         Serial.print(" elapsed "); Serial.print(elapsed);
02484         Serial.print(" timeout "); Serial.print(timeout);
02485         Serial.println(" millisecs");
02486         return false;

```

```

02487     }
02488     if (timer_running) {
02489         Serial.print("Error: timer_wait oops! Return with timer still running");
02490         Serial.print(" elapsed "); Serial.print(elapsed);
02491         Serial.print(" timeout "); Serial.print(timeout);
02492         Serial.println(" millisecs");
02493         return false;
02494     }
02495
02496     if (verbose) {
02497         Serial.print("Success: timer_wait");
02498         Serial.print(" running "); Serial.print(timer_running);
02499         Serial.print(" error "); Serial.print(error_flag);
02500         Serial.print(" elapsed "); Serial.print(elapsed);
02501         Serial.print(" timeout "); Serial.print(timeout);
02502         Serial.println(" millisecs");
02503     }
02504
02505     return true;
02506 }
02507
02508 bool setup_timer(float exposure_secs, float exposure_offset_secs, unsigned int ncounts=0)
02509 {
02510     unsigned int u32;
02511     uint16_t cmpf_mask;
02512     char cbuffer[128] = {0};
02513
02514     // stop everything
02515     stop_runs_only(); // this clears run bits but leaves interrupts attached
02516
02517     // we cannot touch the run mode, only clear the trigger mode
02518     trigger_mode = false;
02519
02520     // clear error and oops flags
02521     clear_error_flags();
02522
02523     // reset elapsed time counters
02524     timer_cyccnt64_start = 0;
02525     timer_cyccnt64_prev = 0;
02526     timer_cyccnt64_now = 0;
02527
02528     // Check, timer is for single pulse flexpwm only
02529     if ((mode!=PULSE) && (mode!=TIMER)) {
02530         Serial.println("Error: setup_timer, need to call setup_pulse() first");
02531         return false;
02532     }
02533
02534     // Check, minimum exposure is the readout and transfer time
02535     if (exposure_secs < timer_interframe_min_secs) {
02536         sprintf(cbuffer,
02537             "Error: setup_timer exposure %.6gs is too short for this pulse configuration, need at least
%.6g",
02538             exposure_secs,timer_interframe_min_secs);
02539         Serial.println(cbuffer);
02540         return false;
02541     }
02542
02543     // Check, maximum offset is length of one exposure
02544     if (exposure_offset_secs > exposure_secs) {
02545         sprintf(cbuffer,
02546             "Error: setup_timer exposure offset %.6gs is too long, need offset less than exposure",
02547             exposure_offset_secs);
02548         Serial.println(cbuffer);
02549         return false;
02550     }
02551
02552     /* =====
02553     Setup timer callback, counts and period
02554     */
02555     timer_callback = pulse_start;
02556     timer_outer_counts = ncounts ? ncounts+1 : frame_counts;
02557     frame_counts = timer_outer_counts;
02558
02559     // Exposure within one iteration of the counter
02560     if (exposure_secs <= COUNTER_MAX_SECS) {
02561         timer_period_secs = exposure_secs;
02562         timer_interval_secs = exposure_secs;
02563         timer_inner_counts = 1;
02564     }
02565
02566     // Exposure requires multiple iterations, set period greater than transfer and offset

```

```

02567     else {
02568         if (USBTRANSFERSECS > 0.024 || exposure_offset_secs > 0.025) {
02569             timer_period_secs = 0.050;
02570         }
02571         else if (USBTRANSFERSECS > 0.009 || exposure_offset_secs > 0.010) {
02572             timer_period_secs = 0.025;
02573         }
02574         else {
02575             timer_period_secs = 0.010;
02576         }
02577         timer_inner_counts = 0;
02578     }
02579
02580     /* =====
02581        From here we setup the control structure and hardware
02582    */
02583     timer.prescale = 0;
02584     timer.divider = 1;
02585     u32 = (timer_period_secs*F_BUS_ACTUAL)/timer.divider;
02586     while (u32 > 65535 && timer.prescale < 7) {
02587         timer.prescale++;
02588         timer.divider = (1<<timer.prescale);
02589         u32 = (timer_period_secs*F_BUS_ACTUAL)/timer.divider;
02590     }
02591
02592     if (u32 > 65535) {
02593         Serial.println("Error: unable to find prescale divider for exposure time");
02594         return false;
02595     }
02596
02597     timer.period_counts = u32;
02598
02599     timer_period_secs = (float) timer.period_counts * timer.divider / F_BUS_ACTUAL;
02600
02601     // round to nearest multiple of timer_period_secs
02602     if (!timer_inner_counts) {
02603         timer_inner_counts = floor((exposure_secs+timer_period_secs/2)/timer_period_secs);
02604     }
02605
02606     timer_interval_secs = timer_period_secs * timer_inner_counts;
02607
02608     Serial.print("timer: timer inner period secs "); Serial.print(timer_period_secs,6);
02609     Serial.print(" counts "); Serial.print(timer_inner_counts);
02610     Serial.print(" actual interval "); Serial.println(timer_interval_secs,6);
02611
02612     timer.onA_counts = 0;
02613     timer.offA_counts = 0;
02614     timer.invertA = false;
02615
02616     timer.onB_counts = 0;
02617     timer.offB_counts = ceil((exposure_offset_secs*F_BUS_ACTUAL)/timer.divider);
02618     cmpf_mask = CMPF_MASKB_OFF;
02619     if (!timer.offB_counts) {
02620         timer.offB_counts = 1;
02621         cmpf_mask = CMPF_MASKB_ON;
02622     }
02623     timer.invertB = false;
02624
02625     timer.inten_mask = 0;
02626
02627     timer.ctrl2_mask = PWM_CTRL2_CLOCK_MASTER;
02628
02629     Serial.println("");
02630     if (!print_and_check_submodule(&timer)) {
02631         return false;
02632     }
02633
02634     // =====
02635     // timer mode is enabled
02636     mode = TIMER; // we only have timer+pulse, we do not have timer+frameset
02637
02638     // setup the hardware
02639     load_submodule(&timer);
02640
02641     // and, attach to the timer irq
02642     attach_isr( &timer, cmpf_mask, timer_isr);
02643
02644     return true;
02645 }
02646
02647 /* =====

```

```

02648     Setup triggers
02649     */
02650     //#define DEBUG_TRIGGERS
02651
02652     inline static bool trigger_busy = false;
02653
02654     static void trigger_isr()
02655     {
02656         uint64_t cyccnt_now = cycles64();
02657
02658     #ifdef DEBUG_TRIGGERS
02659         Serial.print("interrupt "); Serial.println(trigger_counter);
02660         Serial.print(" t="); Serial.println( (float)(cyccnt_now-interrupt_cyccnt64_start)/F_CPU, 6);
02661     #endif
02662
02663         if (trigger_busy) {
02664             oops_flag = true;
02665             Serial.println("Oops! trigger interrupt while busy");
02666             return;
02667         }
02668         trigger_busy = true;
02669
02670         // need to do this before sending data to the host
02671         trigger_cyccnt64_prev = trigger_cyccnt64_now;
02672         trigger_cyccnt64_now = cyccnt_now;
02673
02674         // the callback should have the right counter
02675         trigger_counter++;
02676
02677         // this might result in a data send
02678         if (trigger_callback) {
02679             trigger_callback();
02680         }
02681
02682         // are we stopping?
02683         if (trigger_counter >= trigger_counts) {
02684             detachInterrupt(digitalPinToInterrupt(trigger_pin));
02685             trigger_attached = false;
02686
02687     #ifdef DEBUG_TRIGGERS
02688         Serial.println("interrupt stop");
02689     #endif
02690         }
02691
02692         trigger_busy = false;
02693     }
02694
02695     static void stop_triggers()
02696     {
02697         if (trigger_attached) {
02698             detachInterrupt(digitalPinToInterrupt(trigger_pin));
02699             trigger_attached = false;
02700         }
02701     }
02702
02703     bool start_triggers( )
02704     {
02705         oops_flag = false;
02706         error_flag = false;
02707
02708         if (!trigger_mode) {
02709             Serial.println("start triggers but not trigger mode");
02710             return false;
02711         }
02712         if (trigger_attached) {
02713             Serial.println("start triggers but trigger already attached");
02714             return false;
02715         }
02716
02717         Serial.printf("start_triggers %d\n", trigger_pin);
02718
02719         trigger_counter = 0;
02720
02721         trigger_cyccnt64_start = cycles64();
02722         trigger_cyccnt64_now = trigger_cyccnt64_start;
02723         trigger_cyccnt64_prev = trigger_cyccnt64_start;
02724
02725         // attach the trigger interrupt
02726         trigger_attached = true;
02727         attachInterrupt(digitalPinToInterrupt(trigger_pin), trigger_isr, trigger_edge_mode);
02728

```

```

02729     return true;
02730 }
02731
02732 bool wait_triggers(float timeout=1., float timestep=0.01, bool verbose=false)
02733 {
02734     uint32_t timeout = (uint) (timeout_ * 1000);
02735     uint32_t increment = (uint) (timestep_ * 1000);
02736     uint32_t elapsed = 0;
02737
02738     if (verbose) {
02739         Serial.printf("trigger_wait %d %d\n", timeout, increment);
02740     }
02741
02742     while(trigger_attached && !error_flag && elapsed < timeout) {
02743         delay(increment);
02744         elapsed += increment;
02745     }
02746     if (error_flag) {
02747         Serial.println("Error: trigger_wait error_flag is set.");
02748         return false;
02749     }
02750     if (elapsed >= timeout) {
02751         Serial.printf("Error: wait_interrupt timedout %d / %d msecs\n",elapsed,timeout );
02752         return false;
02753     }
02754     if (timer_running) {
02755         Serial.printf("Error: timer_wait oops! still running %d / %d msecs\n",elapsed,timeout);
02756         return false;
02757     }
02758
02759     if (verbose) {
02760         Serial.printf("Success: timer_wait running %x error %x %d/%d msecs\n",
02761             trigger_attached, error_flag, elapsed, timeout);
02762     }
02763
02764     return true;
02765 }
02766
02767 bool setup_triggers(unsigned int ncounts=0)
02768 {
02769     {
02770         stop_runs_only();    // clear the run bits
02771
02772         // we cannot clear the operating mode, only the trigger mode.
02773         trigger_mode = false;
02774         error_flag = false;
02775         oops_flag = false;
02776
02777         if (mode==NOTCONFIGURED) {
02778             Serial.println("Error: setup_interrupt, but flexpwm is not configured, call setup_flexpwm() first");
02779             error_flag = true;
02780             return false;
02781         }
02782
02783         else if (mode==PULSE) {
02784
02785             trigger_callback = pulse_start;
02786             trigger_counts   = ncounts ? ncounts : frame_counts;
02787
02788             if(!trigger_counts) {
02789                 Serial.println("Error: setup_interrupt pulse, but no frame_counts");
02790                 error_flag = true;
02791                 return false;
02792             }
02793
02794             frame_counts     = trigger_counts;
02795             trigger_mode = true;
02796         }
02797
02798         else if (mode==TIMER) {
02799             trigger_callback = timer_start;
02800             trigger_counts   = ncounts ? ncounts : 1;
02801             trigger_mode = true;
02802         }
02803
02804         else if (mode==FRAMESET) {
02805
02806             trigger_callback = frameset_start;
02807             trigger_counts   = ncounts ? ncounts : frameset_counts;
02808
02809             if(!trigger_counts) {

```

```

02810         Serial.println("Error: setup_interrupt frameset, but no frameset_counts");
02811         error_flag = true;
02812         return false;
02813     }
02814
02815     frameset_counts = trigger_counts;
02816     trigger_mode = true;
02817 }
02818
02819 else {
02820     Serial.println("Error: setup_interrupt, but mode not recognized");
02821     error_flag = true;
02822 }
02823
02824 return trigger_mode;
02825 }
02826
02827
02831 /*
02832 IMXRT_FLEXPWM_t *q = p->flexpwm;
02833 unsigned int submod = p->submod;
02834 uint16_t mask = p->mask;
02835 */
02836
02837 void set_clock_master(uint8_t submod, IMXRT_FLEXPWM_t *q=flexpwm)
02838 {
02839     q->SM[submod].CTRL2 &= 0xFFF0;
02840     q->SM[submod].CTRL2 |= PWM_CTRL2_CLOCK_MASTER;
02841 }
02842
02843 void set_clock_master(SubModule *p)
02844 {
02845     set_clock_master(p->submod, p->flexpwm);
02846 }
02847
02848 // -----
02849 void set_clock_slave(uint8_t submod, IMXRT_FLEXPWM_t *q=flexpwm)
02850 {
02851     q->SM[submod].CTRL2 &= 0xFFF0;
02852     q->SM[submod].CTRL2 |= PWM_CTRL2_CLOCK_SLAVE;
02853 }
02854
02855 void set_clock_slave(SubModule *p)
02856 {
02857     set_clock_slave(p->submod, p->flexpwm);
02858 }
02859
02860 // -----
02861 void set_clock_sync(uint8_t submod, IMXRT_FLEXPWM_t *q=flexpwm) {
02862     q->SM[submod].CTRL2 &= 0xFFF0;
02863     q->SM[submod].CTRL2 |= PWM_CTRL2_CLOCK_SYNC;
02864 }
02865
02866 void set_clock_sync(SubModule *p)
02867 {
02868     set_clock_sync(p->submod, p->flexpwm);
02869 }
02870
02871 // -----
02872 void clear_ldok(uint8_t mask, IMXRT_FLEXPWM_t *q=flexpwm) {
02873     q->MCTRL |= FLEXPWM_MCTRL_CLDOK(mask);
02874 }
02875
02876 void set_ldok(uint8_t mask, IMXRT_FLEXPWM_t *q=flexpwm) {
02877     q->MCTRL |= FLEXPWM_MCTRL_LDOK(mask);
02878 }
02879
02880 // -----
02881 void clear_run(uint8_t mask, IMXRT_FLEXPWM_t *q=flexpwm) {
02882     q->MCTRL &= FLEXPWM_MCTRL_RUN(~mask);
02883 }
02884 void set_run(uint8_t mask, IMXRT_FLEXPWM_t *q=flexpwm) {
02885     q->MCTRL |= FLEXPWM_MCTRL_RUN(mask);
02886 }
02887
02888 // -----
02889 void force(uint8_t submod, IMXRT_FLEXPWM_t *q=flexpwm) {
02890     if(submod<4) {
02891         q->SM[submod].CTRL2 = FLEXPWM_SMCTRL2_FORCE;
02892     }
02893 }

```



```

02894
02895 // -----
02896 void set_prescale(uint8_t submod, uint8_t prescale=0, IMXRT_FLEXPWM_t *q=flexpwm) {
02897     if (submod<4) {
02898         q->SM[submod].CTRL = FLEXPWM_SMCTRL_FULL | FLEXPWM_SMCTRL_PRSC(prescale);
02899     }
02900 }
02901
02902 void set_init(uint8_t submod, uint16_t value, IMXRT_FLEXPWM_t *q=flexpwm) {
02903     if (submod<4) {
02904         q->SM[submod].INIT = value;
02905     }
02906 }
02907
02908 void set_val0(uint8_t submod, uint16_t value, IMXRT_FLEXPWM_t *q=flexpwm) {
02909     if (submod<4) {
02910         q->SM[submod].VAL0 = value;
02911     }
02912 }
02913
02914 void set_val1(uint8_t submod, uint16_t value, IMXRT_FLEXPWM_t *q=flexpwm) {
02915     if (submod<4) {
02916         q->SM[submod].VAL1 = value;
02917     }
02918 }
02919
02920 void set_val2(uint8_t submod, uint16_t value, IMXRT_FLEXPWM_t *q=flexpwm) {
02921     if (submod<4) {
02922         q->SM[submod].VAL2 = value;
02923     }
02924 }
02925
02926 void set_val3(uint8_t submod, uint16_t value, IMXRT_FLEXPWM_t *q=flexpwm) {
02927     if (submod<4) {
02928         q->SM[submod].VAL3 = value;
02929     }
02930 }
02931
02932 void set_val4(uint8_t submod, uint16_t value, IMXRT_FLEXPWM_t *q=flexpwm) {
02933     if (submod<4) {
02934         q->SM[submod].VAL4 = value;
02935     }
02936 }
02937
02938 void set_val5(uint8_t submod, uint16_t value, IMXRT_FLEXPWM_t *q=flexpwm) {
02939     if (submod<4) {
02940         q->SM[submod].VAL5 = value;
02941     }
02942 }
02943 // -----
02944 uint16_t set_outen( uint16_t mask16, IMXRT_FLEXPWM_t *q=flexpwm) {
02945     q->OUTEN = mask16;
02946     return q->OUTEN;
02947 }
02948
02949 uint16_t set_outen_on( uint16_t mask16, IMXRT_FLEXPWM_t *q=flexpwm) {
02950     q->OUTEN |= mask16;
02951     return q->OUTEN;
02952 }
02953
02954 uint16_t set_outen_off( uint16_t mask16, IMXRT_FLEXPWM_t *q=flexpwm) {
02955     q->OUTEN &= ~mask16;
02956     return q->OUTEN;
02957 }
02958
02959 uint16_t set_outenA_on( uint8_t mask8, IMXRT_FLEXPWM_t *q=flexpwm) {
02960     q->OUTEN |= FLEXPWM_OUTEN_PWMA_EN(mask8);
02961     return q->OUTEN;
02962 }
02963
02964
02965
02966 // =====
02967
02968 void printbits16_(uint16_t u16, int bits)
02969 {
02970     while(bits> 0) {
02971         bits--;
02972         Serial.print((u16>>bits)&1);
02973     }
02974 }

```

```

02975
02976 void register_dump(IMXRT_FLEXPWM_t * const q=flexpwm, uint8_t mask=0xFF)
02977 {
02978     uint16_t u16;
02979
02980     Serial.print("flexpwm ");
02981     Serial.println((unsigned int)q, HEX);
02982
02983     u16 = q->MCTRL;
02984     Serial.print("MCTRL "); printbits16_(u16,16); // Serial.print(u16,BIN);
02985     Serial.print(" IPOL "); printbits16_((u16>12),4); // Serial.print((u16>12)&0xF,BIN);
02986     Serial.print(" RUN "); printbits16_((u16>8),4); // Serial.print((u16>8)&0xF,BIN);
02987     Serial.print(" LDOK "); printbits16_(u16,4); // Serial.print(u16&0xF,BIN);
02988     Serial.println("");
02989
02990     u16 = q->MCTRL2;
02991     Serial.print("MCTRL2 MONPLL "); Serial.print(u16&0xF,BIN);
02992     Serial.println("");
02993
02994     u16 = q->OUTEN;
02995     Serial.print("OUTEN "); printbits16_(u16,16); // Serial.print(u16,BIN);
02996     Serial.print(" PWMA "); printbits16_((u16>8),4); // Serial.print((u16>8)&0xF,BIN);
02997     Serial.print(" PWMB "); printbits16_((u16>4),4); // Serial.print((u16>4)&0xF,BIN);
02998     Serial.print(" PWMX "); printbits16_(u16,4); // Serial.print(u16&0xF,BIN);
02999     Serial.println("");
03000
03001     for (uint8_t submodule = 0; submodule<4; submodule++) {
03002
03003         if (mask & (1<<submodule)) {
03004
03005             Serial.println("#-----");
03006             Serial.print("Submodule "); Serial.print(submodule);
03007             Serial.println("");
03008
03009             u16 = q->SM[submodule].CNT;
03010             Serial.print("CNT "); Serial.println(u16);
03011
03012             u16 = q->SM[submodule].INIT;
03013             Serial.print("INIT "); Serial.println(u16);
03014
03015             u16 = q->SM[submodule].CTRL2;
03016             Serial.print("CTRL2 "); printbits16_(u16,16); // Serial.print(u16,BIN);
03017             Serial.print(" DBGEN "); Serial.print((u16>15));
03018             Serial.print(" WAITEN "); Serial.print((u16>14)&1);
03019             Serial.print(" INDEP "); Serial.print((u16>13)&1);
03020             Serial.print(" PWM23_INIT "); Serial.print((u16>12)&1);
03021             Serial.print(" PWM45_INIT "); Serial.print((u16>11)&1);
03022             Serial.print(" PWMX_INIT "); Serial.print((u16>10)&1);
03023             Serial.print(" INIT_SEL "); Serial.print((u16>8)&3);
03024             Serial.print(" FRCEN "); Serial.print((u16>7)&1);
03025             Serial.print(" FORCE "); Serial.print((u16>6)&1);
03026             Serial.print(" FORCE_SEL "); Serial.print((u16>3)&7);
03027             Serial.print(" RELOAD_SEL "); Serial.print((u16>2)&1);
03028             Serial.print(" CLK_SEL "); Serial.print((u16>3));
03029             Serial.println("");
03030
03031             u16 = q->SM[submodule].CTRL;
03032             Serial.print("CTRL "); printbits16_(u16,16); // Serial.print(u16,BIN);
03033             Serial.print(" LDFQ "); Serial.print((u16>12)&0xF);
03034             Serial.print(" HALF "); Serial.print((u16>11)&1);
03035             Serial.print(" FULL "); Serial.print((u16>10)&1);
03036             Serial.print(" DT "); Serial.print((u16>9)&0x1); Serial.print((u16>8)&0x1);
03037             Serial.print(" COMPMODE "); Serial.print((u16>10)&1);
03038             Serial.print(" PRSC "); Serial.print((u16>4)&0x7);
03039             Serial.print(" SPLIT "); Serial.print((u16>3)&1);
03040             Serial.print(" LDMOD "); Serial.print((u16>2)&1);
03041             Serial.print(" DBLX "); Serial.print((u16>1)&1);
03042             Serial.print(" DBLEN "); Serial.print((u16>1));
03043             Serial.println("");
03044
03045             u16 = q->SM[submodule].OCTRL;
03046             Serial.print("OCTRL "); printbits16_(u16,16); // Serial.print(u16,BIN);
03047             Serial.print(" PWMA_IN "); Serial.print((u16>15)&1);
03048             Serial.print(" PWMB_IN "); Serial.print((u16>14)&1);
03049             Serial.print(" PWMX_IN "); Serial.print((u16>13)&1);
03050             Serial.print(" POLA "); Serial.print((u16>10)&1);
03051             Serial.print(" POLB "); Serial.print((u16>9)&1);
03052             Serial.print(" POLX "); Serial.print((u16>8)&1);
03053             Serial.print(" PWMA_FS "); Serial.print((u16>4)&3);
03054             Serial.print(" PWMB_FS "); Serial.print((u16>2)&3);
03055             Serial.print(" PWMX_FS "); Serial.print(u16>3);

```

```

03056     Serial.println("");
03057
03058     u16 = q->SM[submodule].VAL0;
03059     Serial.print("VAL0 "); Serial.print(u16);
03060
03061     u16 = q->SM[submodule].VAL1;
03062     Serial.print(" VAL1 "); Serial.print(u16);
03063
03064     u16 = q->SM[submodule].VAL2;
03065     Serial.print(" VAL2 "); Serial.print(u16);
03066
03067     u16 = q->SM[submodule].VAL3;
03068     Serial.print(" VAL3 "); Serial.print(u16);
03069
03070     u16 = q->SM[submodule].VAL4;
03071     Serial.print(" VAL4 "); Serial.print(u16);
03072
03073     u16 = q->SM[submodule].VAL5;
03074     Serial.print(" VAL5 "); Serial.print(u16);
03075     Serial.println("");
03076
03077     u16 = q->SM[submodule].STS;
03078     Serial.print("STS "); printbits16_(u16,16); // Serial.print(u16,BIN);
03079     Serial.print(" RUF "); Serial.print(((u16>14)&0x1));
03080     Serial.print(" REF "); Serial.print(((u16>13)&0x1));
03081     Serial.print(" RF "); Serial.print(((u16>12)&0x1));
03082     Serial.print(" CFA1 "); Serial.print(((u16>11)&0x1));
03083     Serial.print(" CFA0 "); Serial.print(((u16>10)&0x1));
03084     Serial.print(" CFB1 "); Serial.print(((u16>9)&0x1));
03085     Serial.print(" CFB0 "); Serial.print(((u16>8)&0x1));
03086     Serial.print(" CFX1 "); Serial.print(((u16>7)&0x1));
03087     Serial.print(" CFX0 "); Serial.print(((u16>6)&0x1));
03088     Serial.print(" CMPF "); printbits16_(u16,6); // Serial.print(u16&0x3F,BIN);
03089     Serial.println("");
03090
03091     u16 = q->SM[submodule].INTEN;
03092     Serial.print("INTEN "); printbits16_(u16,16); // Serial.print(u16,BIN);
03093     Serial.print(" REIE "); Serial.print(((u16>13)&0x1));
03094     Serial.print(" RIE "); Serial.print(((u16>12)&0x1));
03095     Serial.print(" CALIE "); Serial.print(((u16>11)&0x1));
03096     Serial.print(" CA0IE "); Serial.print(((u16>10)&0x1));
03097     Serial.print(" CB1IE "); Serial.print(((u16>9)&0x1));
03098     Serial.print(" CB0IE "); Serial.print(((u16>8)&0x1));
03099     Serial.print(" CX1IE "); Serial.print(((u16>7)&0x1));
03100     Serial.print(" CX0IE "); Serial.print(((u16>6)&0x1));
03101     Serial.print(" CMPIE "); printbits16_(u16,6); // Serial.print(u16&0x3F,BIN);
03102     Serial.println("");
03103 }
03104 }
03105 }
03106
03107 // -----
03108 static void setup_digital_pins()
03109 {
03110     pinMode(TRIGGER_PIN, INPUT);
03111
03112     pinMode(BUSY_PIN, OUTPUT);
03113     digitalWriteFast(BUSY_PIN, BUSY_PIN_DEFAULT);
03114
03115     pinMode(SYNC_PIN, OUTPUT);
03116     digitalWriteFast(SYNC_PIN, SYNC_PIN_DEFAULT);
03117
03118     // Clock
03119     pinMode(CLK_PIN, OUTPUT);
03120     digitalWriteFast(CLK_PIN, LOW);
03121
03122     pinMode(CLK_MONITOR_PIN, INPUT);
03123
03124     // SH gate
03125     pinMode(SH_PIN, OUTPUT);
03126     digitalWriteFast(SH_PIN, LOW);
03127
03128     // Integratinon Clear Gate
03129     pinMode(ICG_PIN, OUTPUT);
03130     digitalWriteFast(ICG_PIN, HIGH);
03131
03132     // CNVST to ADC
03133     pinMode(CNVST_PIN, OUTPUT);
03134     digitalWriteFast(CNVST_PIN, LOW);
03135
03136     // clear the mode

```

```

03137     clear_mode();
03138 }
03139
03140 static void close()
03141 {
03142     // stop both flexpwms, disable interrupts and reset clear and busy pins
03143     stop_with_irqs();
03144
03145     // Reactivation as digital i/o pins
03146     setup_digital_pins();
03147 }
03148
03149
03150
03151
03152 };
03153
03154 #endif

```

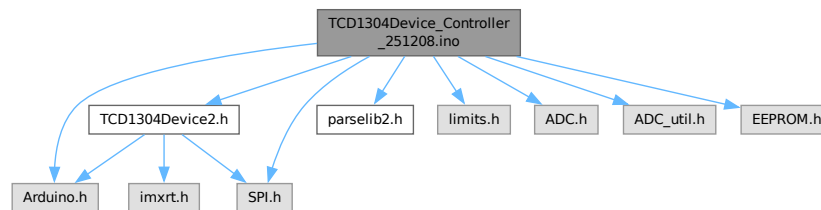
## 5.7 TCD1304Device\_Controller\_251208.ino File Reference

```

#include "Arduino.h"
#include "TCD1304Device2.h"
#include "parselib2.h"
#include <SPI.h>
#include <limits.h>
#include <ADC.h>
#include <ADC_util.h>
#include <EEPROM.h>

```

Include dependency graph for TCD1304Device\_Controller\_251208.ino:



### Classes

- struct [usb\\_string\\_descriptor\\_struct\\_manufacturer](#)
- struct [usb\\_string\\_descriptor\\_struct\\_product](#)
- struct [usb\\_string\\_descriptor\\_struct\\_serial\\_number](#)

### Macros

- #define [DRMCNELSONLAB](#)
- #define [CONTROLLER\\_OCPIN](#) 23
- #define [CONTROLLER\\_CSPIN](#) 10
- #define [CONTROLLER\\_CSPIN2](#) 9

- #define [CYCLES\\_PER\\_USEC](#) (F\_CPU / 1000000)
- #define [thisMANUFACTURER\\_NAME](#) {'D','R','M','C','N','E','L','S','O','N','L','A','B' }
- #define [thisMANUFACTURER\\_NAME\\_LEN](#) 13
- #define [thisPRODUCT\\_NAME](#) {'T','C','D','1','3','0','4','S','P','I'}
- #define [thisPRODUCT\\_NAME\\_LEN](#) 10
- #define [thisPRODUCT\\_SERIAL\\_NUMBER](#) { 'S','N','0','0','0','0','0','0','0','0','0','1' }
- #define [thisPRODUCT\\_SERIAL\\_NUMBER\\_LEN](#) 12
- #define [RCVLEN](#) 256
- #define [SNDLEN](#) 256
- #define [IMR\\_INDEX](#) 5
- #define [ISR\\_INDEX](#) 6
- #define [NADC\\_CHANNELS](#) 4
- #define [ADC\\_RESOLUTION](#) (3.3/4096.)
- #define [EEPROM\\_SIZE](#) 1080
- #define [EEPROM\\_ID\\_ADDR](#) 0
- #define [EEPROM\\_ID\\_LEN](#) 64
- #define [EEPROM\\_COEFF\\_ADDR](#) 64
- #define [EEPROM\\_NCOEFFS](#) 4
- #define [EEPROM\\_COEFF\\_LEN](#) (EEPROM\_NCOEFFS \* sizeof(float))
- #define [EEPROM\\_UNITS\\_ADDR](#) (EEPROM\_COEFF\_ADDR + EEPROM\_COEFF\_LEN)
- #define [EEPROM\\_NUNITS](#) 8
- #define [EEPROM\\_UNITS\\_LEN](#) EEPROM\_NUNITS
- #define [NBUFFERS](#) 32
- #define [BINARY](#) 0
- #define [ASCII](#) 1

## Functions

- SPISettings [spi\\_settings](#) (30000000, MSBFIRST, SPI\_MODE0)
- float [tempmonGetTemp](#) (void)
- void [blink](#) ()
- void [pulsePin](#) (unsigned int pin, unsigned int usecs)
- void [ocpinRead](#) ()
- void [ocpinISR](#) ()
- void [ocpinAttach](#) ()
- void [ocpinDetach](#) ()
- void [resumePinInterrupts](#) (uint8\_t pin)
- void [disablePinInterrupts](#) (uint8\_t pin)
- uint64\_t [cycles64](#) ()
- int [fastAnalogRead](#) (uint8\_t pin)
- void [sendADCs](#) (unsigned int averages)
- void [sendChipTemperature](#) (unsigned int averages)
- void [eeread](#) (unsigned int address, int nbytes, char \*p)
- void [eewrite](#) (unsigned int address, int nbytes, char \*p)
- void [eereadUntil](#) (unsigned int address, int nbytes, char \*p)
- void [eewriteUntil](#) (unsigned int address, int nbytes, char \*p)
- void [eeErase](#) (unsigned int address, int nbytes)
- void [eraselIdentifier](#) ()
- void [readIdentifier](#) (char \*p)
- void [storeIdentifier](#) (char \*p)

- void `printIdentifier` ()
- void `eraseUnits` ()
- void `readUnits` (char \*p)
- void `storeUnits` (char \*p)
- void `printUnits` ()
- void `eraseCoefficients` ()
- void `readCoefficients` (float \*vals)
- void `storeCoefficients` (float \*vals)
- void `printCoefficients` ()
- void `clock_isr` ()
- void `clock_stop` ()
- void `clock_start` ()
- bool `clock_setup` (float secs, unsigned int ncounts=0)
- uint8\_t `calculateCRC` (uint16\_t \*bufferp)
- uint32\_t `sumData` (uint16\_t \*bufferp)
- void `sendBuffer_Formatted` (uint16\_t \*bp)
- void `sendBuffer_Binary` (uint16\_t \*bp)
- void `sendDataCRC` (uint16\_t \*bp)
- void `sendDataSum` (uint16\_t \*bp)
- void `sendData` (uint16\_t \*bp)
- void `sendDataReady` ()
- void `sendTestData` (uint16\_t \*bp)
- void `send_header` ()
- bool `send_frame` (TCD1304Device ::Frame\_Header \*p)
- void `send_frames` ()
- void `frame_callback` ()
- bool `strPin` (char \*pc, uint8\_t \*pin, char \*\*next)
- void `printTriggerSetup` ()
- bool `strTrigger` (char \*pc, char \*\*next)
- bool `strSubModule` (char \*s, char \*\*next=0)
- void `help` (const char \*key)
- void `setup` ()
- void `loop` ()

## Variables

- ADC \* `adc` = new ADC( )
- TCD1304Device `tcd1304device` = TCD1304Device( )
- usb\_string\_descriptor\_struct `manufacturer` `usb_string_manufacturer_name`
- usb\_string\_descriptor\_struct `product` `usb_string_product_name`
- usb\_string\_descriptor\_struct `serial_number` `usb_string_serial_number`
- char `rcvbuffer` [RCVLEN]
- uint16\_t `nrcvbuf` = 0
- char `sndbuffer` [SNDLEN]
- const char `versionstr` [] = "TCD1304Device vers 0.4 " \_\_DATE\_\_
- const char `authorstr` [] = "Patents Pending and (c) 2020, 2023, 2025 by Mitchell C. Nelson, Ph.D. "
- const char `srcfilestr` [] = \_\_FILE\_\_
- const char `sensorstr` [] = "TCD1304"
- const int `syncPin` = SYNC\_PIN
- const int `busyPin` = BUSY\_PIN

- const int `triggerPin` = `TRIGGER_PIN`
- const int `fMPin` = `CLK_PIN`
- const int `ICGPin` = `ICG_PIN`
- const int `SHPin` = `SH_PIN`
- const int `CNVSTPin` = `CNVST_PIN`
- const int `SDIPin` = 11
- const int `SDOPin` = 12
- const int `CLKPin` = 13
- const int `analogPin` = A0
- bool `ocpinstate` = false
- bool `ocpinattached` = false
- elapsedMicros `elapsed_usecs`
- elapsedMicros `diagnostic_usecs`
- bool `diagnostics` = false
- uint16\_t `adc_data` [`NADC_CHANNELS`] = { 0 }
- unsigned int `adc_averages` = 0
- unsigned int `chipTemp_averages` = 0
- IntervalTimer `clock_timer`
- void(\* `clock_callback` )() = nullptr
- unsigned int `clock_counts` = 0
- unsigned int `clock_counter` = 0
- float `clock_usecs` = 0
- bool `clock_mode` = false
- unsigned char `crctable` [256]
- `TCD1304Device::Frame_Header` `frame_header_ring` [`NBUFFERS`] = {0}
- `TCD1304Device::Frame_Header` \* `framep` = &`frame_header_ring`[0]
- uint16\_t `buffer_ring` [`NBUFFERS`][`NREADOUT`] = { 0 }
- uint16\_t \* `bufferp` = `buffer_ring`[0]
- unsigned int `frame_index` = 0
- unsigned int `buffer_index` = 0
- unsigned int `frame_send_index` = 0
- unsigned int `dataformat` = `BINARY`
- unsigned int `counter` = 0
- bool `data_async` = true
- bool `crc_enable` = false
- bool `sum_enable` = false

## 5.7.1 Macro Definition Documentation

### 5.7.1.1 ADC\_RESOLUTION

```
#define ADC_RESOLUTION (3.3/4096.)
```

### 5.7.1.2 ASCII

```
#define ASCII 1
```

### 5.7.1.3 BINARY

```
#define BINARY 0
```

### 5.7.1.4 CONTROLLER\_CSPIN

```
#define CONTROLLER_CSPIN 10
```

### 5.7.1.5 CONTROLLER\_CSPIN2

```
#define CONTROLLER_CSPIN2 9
```

### 5.7.1.6 CONTROLLER\_OCPIN

```
#define CONTROLLER_OCPIN 23
```

### 5.7.1.7 CYCLES\_PER\_USEC

```
#define CYCLES_PER_USEC (F_CPU / 1000000)
```

### 5.7.1.8 DRMCNELSONLAB

```
#define DRMCNELSONLAB
```

### 5.7.1.9 EEPROM\_COEFF\_ADDR

```
#define EEPROM_COEFF_ADDR 64
```

### 5.7.1.10 EEPROM\_COEFF\_LEN

```
#define EEPROM_COEFF_LEN (EEPROM_NCOEFFS * sizeof(float))
```

### 5.7.1.11 EEPROM\_ID\_ADDR

```
#define EEPROM_ID_ADDR 0
```

### 5.7.1.12 EEPROM\_ID\_LEN

```
#define EEPROM_ID_LEN 64
```



#### 5.7.1.13 EEPROM\_NCOEFFS

```
#define EEPROM_NCOEFFS 4
```

#### 5.7.1.14 EEPROM\_NUNITS

```
#define EEPROM_NUNITS 8
```

#### 5.7.1.15 EEPROM\_SIZE

```
#define EEPROM_SIZE 1080
```

#### 5.7.1.16 EEPROM\_UNITS\_ADDR

```
#define EEPROM_UNITS_ADDR (EEPROM_COEFF_ADDR + EEPROM_COEFF_LEN)
```

#### 5.7.1.17 EEPROM\_UNITS\_LEN

```
#define EEPROM_UNITS_LEN EEPROM_NUNITS
```

#### 5.7.1.18 IMR\_INDEX

```
#define IMR_INDEX 5
```

#### 5.7.1.19 ISR\_INDEX

```
#define ISR_INDEX 6
```

#### 5.7.1.20 NADC\_CHANNELS

```
#define NADC_CHANNELS 4
```

#### 5.7.1.21 NBUFFERS

```
#define NBUFFERS 32
```

#### 5.7.1.22 RCVLEN

```
#define RCVLEN 256
```

#### 5.7.1.23 SNDLEN

```
#define SNDLEN 256
```

#### 5.7.1.24 thisMANUFACTURER\_NAME

```
#define thisMANUFACTURER_NAME {'D','R','M','C','N','E','L','S','O','N','L','A','B' }
```

#### 5.7.1.25 thisMANUFACTURER\_NAME\_LEN

```
#define thisMANUFACTURER_NAME_LEN 13
```

#### 5.7.1.26 thisPRODUCT\_NAME

```
#define thisPRODUCT_NAME {'T','C','D','1','3','0','4','S','P','I'}
```

#### 5.7.1.27 thisPRODUCT\_NAME\_LEN

```
#define thisPRODUCT_NAME_LEN 10
```

#### 5.7.1.28 thisPRODUCT\_SERIAL\_NUMBER

```
#define thisPRODUCT_SERIAL_NUMBER { 'S','N','0','0','0','0','0','0','0','0','0','1' }
```

#### 5.7.1.29 thisPRODUCT\_SERIAL\_NUMBER\_LEN

```
#define thisPRODUCT_SERIAL_NUMBER_LEN 12
```

### 5.7.2 Function Documentation

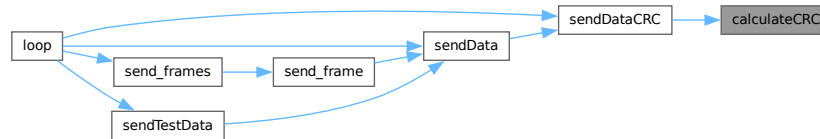
#### 5.7.2.1 blink()

```
void blink ()
```

### 5.7.2.2 calculateCRC()

```
uint8_t calculateCRC (  
    uint16_t * bufferp) [inline]
```

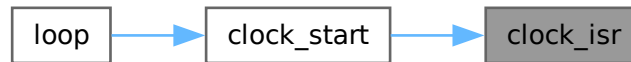
Here is the caller graph for this function:



### 5.7.2.3 clock\_isr()

```
void clock_isr ()
```

Here is the caller graph for this function:



### 5.7.2.4 clock\_setup()

```
bool clock_setup (  
    float secs,  
    unsigned int ncounts = 0)
```

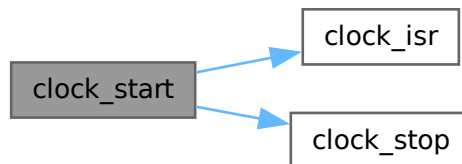
Here is the caller graph for this function:



### 5.7.2.5 clock\_start()

```
void clock_start ()
```

Here is the call graph for this function:



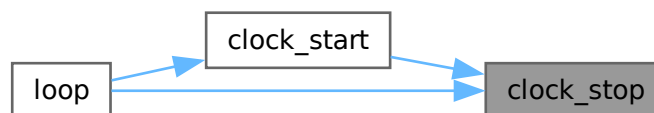
Here is the caller graph for this function:



### 5.7.2.6 clock\_stop()

```
void clock_stop ()
```

Here is the caller graph for this function:



### 5.7.2.7 cycles64()

```
uint64_t cycles64 ()
```

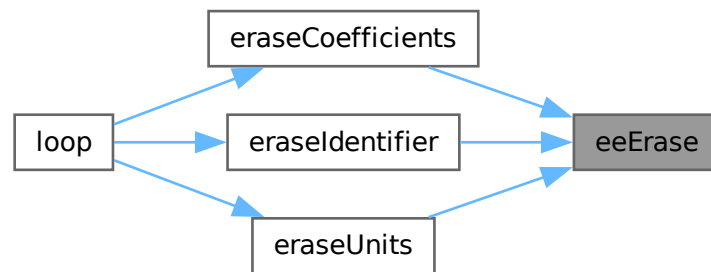
### 5.7.2.8 disablePinInterrupts()

```
void disablePinInterrupts (  
    uint8_t pin)
```

### 5.7.2.9 eeErase()

```
void eeErase (  
    unsigned int address,  
    int nbytes)
```

Here is the caller graph for this function:



### 5.7.2.10 eeread()

```
void eeread (  
    unsigned int address,  
    int nbytes,  
    char * p)
```

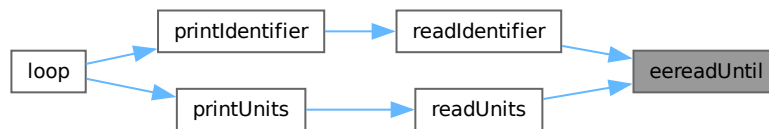
Here is the caller graph for this function:



### 5.7.2.11 eereadUntil()

```
void eereadUntil (  
    unsigned int address,  
    int nbytes,  
    char * p)
```

Here is the caller graph for this function:



### 5.7.2.12 eewrite()

```
void eewrite (  
    unsigned int address,  
    int nbytes,  
    char * p)
```

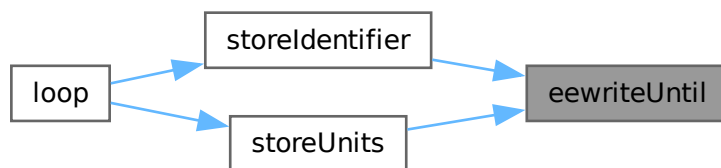
Here is the caller graph for this function:



### 5.7.2.13 eewriteUntil()

```
void eewriteUntil (  
    unsigned int address,  
    int nbytes,  
    char * p)
```

Here is the caller graph for this function:



#### 5.7.2.14 eraseCoefficients()

```
void eraseCoefficients ()
```

Here is the call graph for this function:



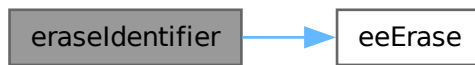
Here is the caller graph for this function:



### 5.7.2.15 eraseIdentifier()

```
void eraseIdentifier ()
```

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.7.2.16 eraseUnits()

```
void eraseUnits ()
```

Here is the call graph for this function:





Here is the caller graph for this function:



#### 5.7.2.17 fastAnalogRead()

```
int fastAnalogRead (  
    uint8_t pin) [inline]
```

Here is the caller graph for this function:



#### 5.7.2.18 frame\_callback()

```
void frame_callback ()
```

Here is the caller graph for this function:



### 5.7.2.19 help()

```
void help (  
    const char * key)
```

Here is the call graph for this function:



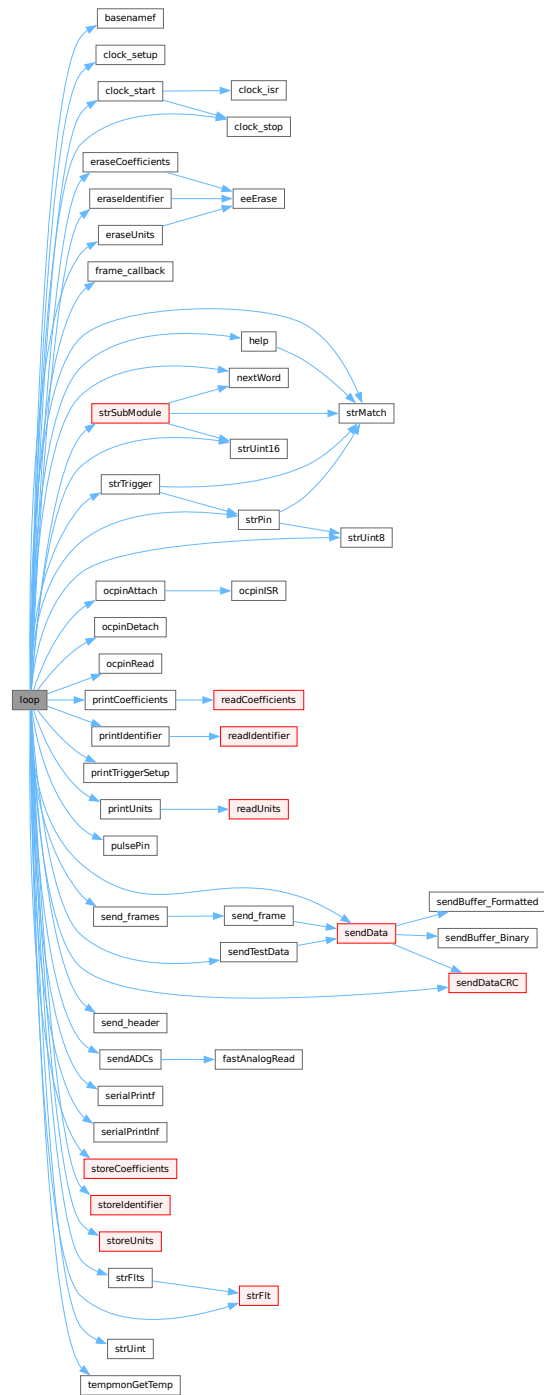
Here is the caller graph for this function:



### 5.7.2.20 loop()

```
void loop ()
```

Here is the call graph for this function:



### 5.7.2.21 ocpinAttach()

```
void ocpinAttach ()
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.7.2.22 ocpinDetach()

```
void ocpinDetach ()
```

Here is the caller graph for this function:



#### 5.7.2.23 ocpinISR()

```
void ocpinISR ()
```

Here is the caller graph for this function:



#### 5.7.2.24 ocpinRead()

```
void ocpinRead ()
```

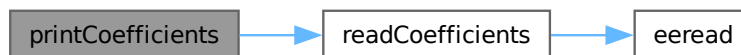
Here is the caller graph for this function:



#### 5.7.2.25 printCoefficients()

```
void printCoefficients ()
```

Here is the call graph for this function:



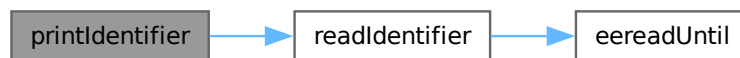
Here is the caller graph for this function:



#### 5.7.2.26 printIdentifier()

```
void printIdentifier ()
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.7.2.27 printTriggerSetup()

```
void printTriggerSetup ()
```

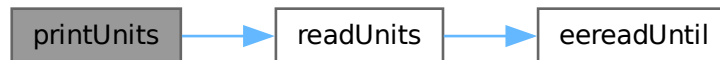
Here is the caller graph for this function:



#### 5.7.2.28 printUnits()

```
void printUnits ()
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.7.2.29 pulsePin()

```
void pulsePin (  
    unsigned int pin,  
    unsigned int usecs)
```

Here is the caller graph for this function:



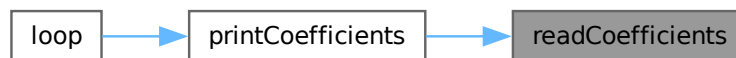
#### 5.7.2.30 readCoefficients()

```
void readCoefficients (  
    float * vals)
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.7.2.31 readIdentifier()

```
void readIdentifier (  
    char * p)
```



Here is the call graph for this function:



Here is the caller graph for this function:



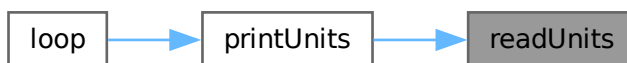
#### 5.7.2.32 readUnits()

```
void readUnits (  
    char * p)
```

Here is the call graph for this function:



Here is the caller graph for this function:



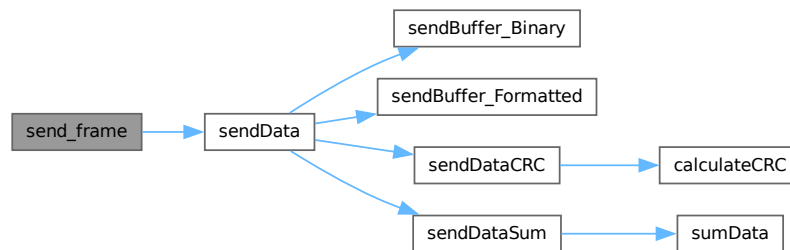
### 5.7.2.33 resumePinInterrupts()

```
void resumePinInterrupts (  
    uint8_t pin)
```

### 5.7.2.34 send\_frame()

```
bool send_frame (  
    TCD1304Device ::Frame_Header * p)
```

Here is the call graph for this function:



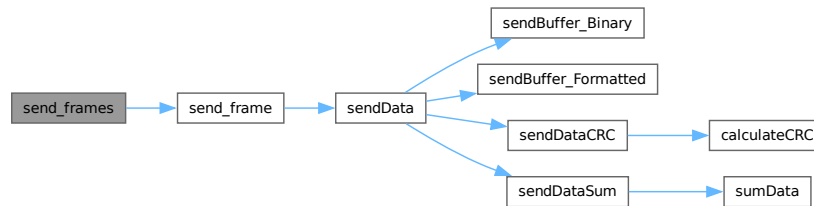
Here is the caller graph for this function:



### 5.7.2.35 send\_frames()

```
void send_frames ()
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.7.2.36 send\_header()

```
void send_header ()
```

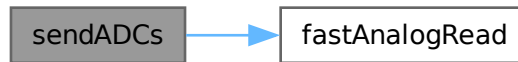
Here is the caller graph for this function:



#### 5.7.2.37 sendADCs()

```
void sendADCs (  
    unsigned int averages)
```

Here is the call graph for this function:



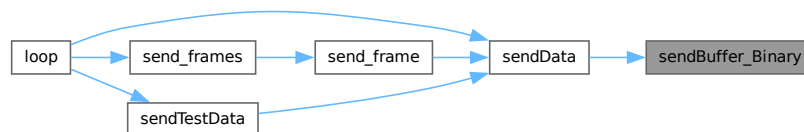
Here is the caller graph for this function:



### 5.7.2.38 sendBuffer\_Binary()

```
void sendBuffer_Binary (
    uint16_t * bp)
```

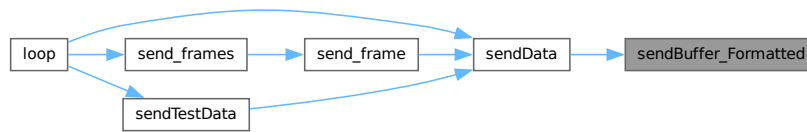
Here is the caller graph for this function:



### 5.7.2.39 sendBuffer\_Formatted()

```
void sendBuffer_Formatted (
    uint16_t * bp)
```

Here is the caller graph for this function:



#### 5.7.2.40 sendChipTemperature()

```
void sendChipTemperature (  
    unsigned int averages)
```

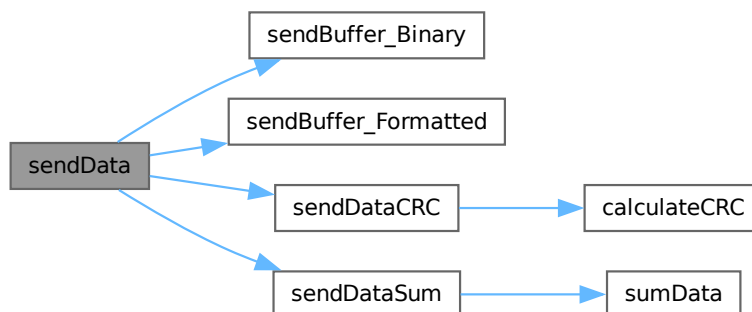
Here is the call graph for this function:



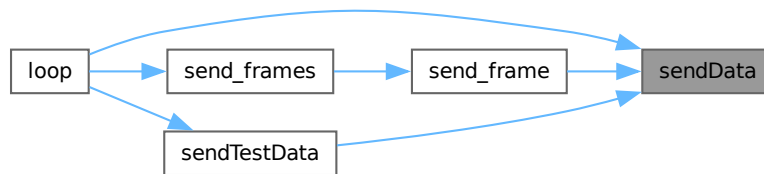
#### 5.7.2.41 sendData()

```
void sendData (  
    uint16_t * bp)
```

Here is the call graph for this function:



Here is the caller graph for this function:



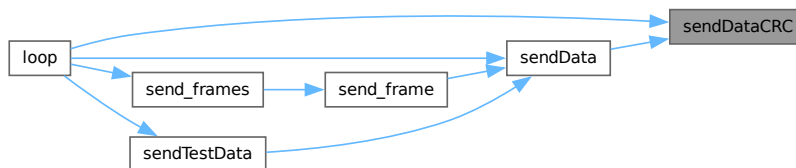
#### 5.7.2.42 sendDataCRC()

```
void sendDataCRC (
    uint16_t * bp) [inline]
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.7.2.43 sendDataReady()

```
void sendDataReady ()
```

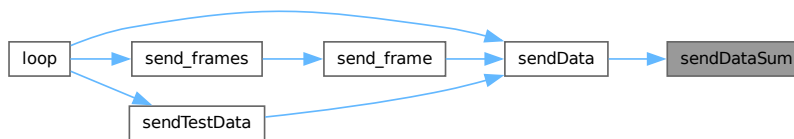
#### 5.7.2.44 sendDataSum()

```
void sendDataSum (  
    uint16_t * bp) [inline]
```

Here is the call graph for this function:



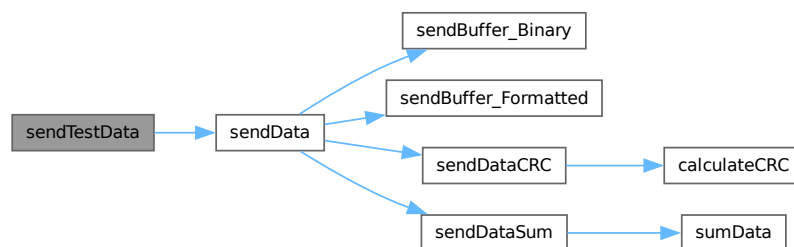
Here is the caller graph for this function:



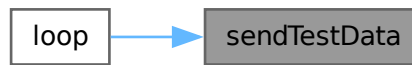
#### 5.7.2.45 sendTestData()

```
void sendTestData (  
    uint16_t * bp)
```

Here is the call graph for this function:



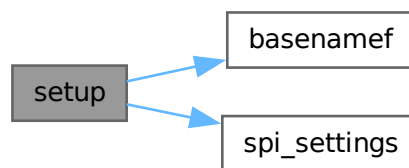
Here is the caller graph for this function:



#### 5.7.2.46 setup()

```
void setup ()
```

Here is the call graph for this function:



#### 5.7.2.47 spi\_settings()

```
SPISettings spi_settings (  
    30000000 ,  
    MSBFIRST ,  
    SPI_MODE0 )
```

Here is the caller graph for this function:

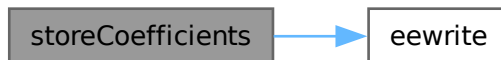




#### 5.7.2.48 storeCoefficients()

```
void storeCoefficients (  
    float * vals)
```

Here is the call graph for this function:



Here is the caller graph for this function:



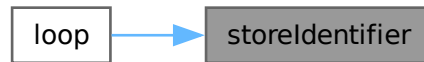
#### 5.7.2.49 storeIdentifier()

```
void storeIdentifier (  
    char * p)
```

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.7.2.50 storeUnits()

```
void storeUnits (  
    char * p)
```

Here is the call graph for this function:



Here is the caller graph for this function:

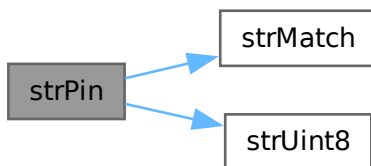


#### 5.7.2.51 strPin()

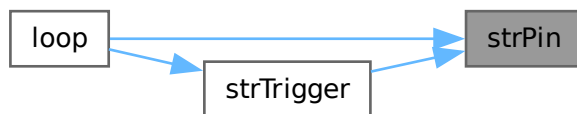
```
bool strPin (  
    char * pc,
```

```
uint8_t * pin,  
char ** next)
```

Here is the call graph for this function:



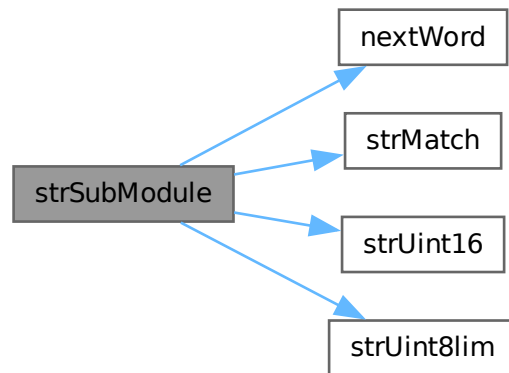
Here is the caller graph for this function:



#### 5.7.2.52 strSubModule()

```
bool strSubModule (  
    char * s,  
    char ** next = 0)
```

Here is the call graph for this function:



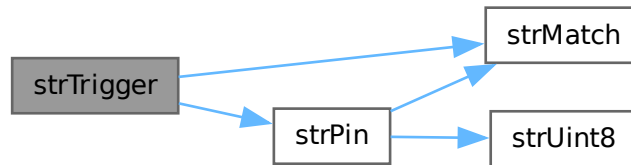
Here is the caller graph for this function:



### 5.7.2.53 strTrigger()

```
bool strTrigger (  
    char * pc,  
    char ** next)
```

Here is the call graph for this function:



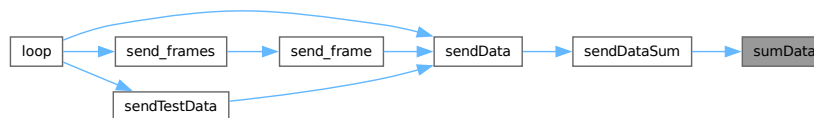
Here is the caller graph for this function:



#### 5.7.2.54 sumData()

```
uint32_t sumData (  
    uint16_t * bufferp) [inline]
```

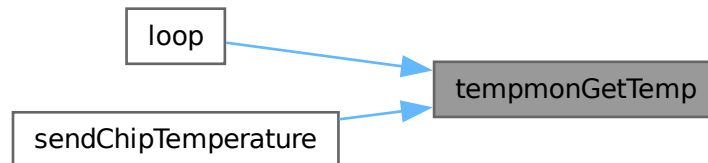
Here is the caller graph for this function:



### 5.7.2.55 tempmonGetTemp()

```
float tempmonGetTemp (
    void ) [extern]
```

Here is the caller graph for this function:



## 5.7.3 Variable Documentation

### 5.7.3.1 adc

```
ADC* adc = new ADC( )
```

### 5.7.3.2 adc\_averages

```
unsigned int adc_averages = 0
```

### 5.7.3.3 adc\_data

```
uint16_t adc_data[NADC_CHANNELS] = { 0 }
```

### 5.7.3.4 analogPin

```
const int analogPin = A0
```

### 5.7.3.5 authorstr

```
const char authorstr[] = "Patents Pending and (c) 2020, 2023, 2025 by Mitchell C. Nelson, Ph.D. "
```

#### 5.7.3.6 buffer\_index

```
unsigned int buffer_index = 0
```

#### 5.7.3.7 buffer\_ring

```
uint16_t buffer_ring[NBUFFERS][NREADOUT] = { 0 }
```

#### 5.7.3.8 bufferp

```
uint16_t* bufferp = buffer_ring[0]
```

#### 5.7.3.9 busyPin

```
const int busyPin = BUSY_PIN
```

#### 5.7.3.10 chipTemp\_averages

```
unsigned int chipTemp_averages = 0
```

#### 5.7.3.11 CLKPin

```
const int CLKPin = 13
```

#### 5.7.3.12 clock\_callback

```
void(* clock_callback) () () = nullptr
```

#### 5.7.3.13 clock\_counter

```
unsigned int clock_counter = 0
```

#### 5.7.3.14 clock\_counts

```
unsigned int clock_counts = 0
```

#### 5.7.3.15 clock\_mode

```
bool clock_mode = false
```

### 5.7.3.16 clock\_timer

```
IntervalTimer clock_timer
```

### 5.7.3.17 clock\_usecs

```
float clock_usecs = 0
```

### 5.7.3.18 CNVSTPin

```
const int CNVSTPin = CNVST_PIN
```

### 5.7.3.19 counter

```
unsigned int counter = 0
```

### 5.7.3.20 crc\_enable

```
bool crc_enable = false
```

### 5.7.3.21 crctable

```
unsigned char crctable[256]
```

#### Initial value:

```
= {  
    0x00, 0x31, 0x62, 0x53, 0xc4, 0xf5, 0xa6, 0x97, 0xb9, 0x88, 0xdb, 0xea, 0x7d, 0x4c, 0x1f, 0x2e,  
    0x43, 0x72, 0x21, 0x10, 0x87, 0xb6, 0xe5, 0xd4, 0xfa, 0xcb, 0x98, 0xa9, 0x3e, 0x0f, 0x5c, 0x6d,  
    0x86, 0xb7, 0xe4, 0xd5, 0x42, 0x73, 0x20, 0x11, 0x3f, 0x0e, 0x5d, 0x6c, 0xfb, 0xca, 0x99, 0xa8,  
    0xc5, 0xf4, 0xa7, 0x96, 0x01, 0x30, 0x63, 0x52, 0x7c, 0x4d, 0x1e, 0x2f, 0xb8, 0x89, 0xda, 0xeb,  
    0x3d, 0x0c, 0x5f, 0x6e, 0xf9, 0xc8, 0x9b, 0xaa, 0x84, 0xb5, 0xe6, 0xd7, 0x40, 0x71, 0x22, 0x13,  
    0x7e, 0x4f, 0x1c, 0x2d, 0xba, 0x8b, 0xd8, 0xe9, 0xc7, 0xf6, 0xa5, 0x94, 0x03, 0x32, 0x61, 0x50,  
    0xbb, 0x8a, 0xd9, 0xe8, 0x7f, 0x4e, 0x1d, 0x2c, 0x02, 0x33, 0x60, 0x51, 0xc6, 0xf7, 0xa4, 0x95,  
    0xf8, 0xc9, 0x9a, 0xab, 0x3c, 0x0d, 0x5e, 0x6f, 0x41, 0x70, 0x23, 0x12, 0x85, 0xb4, 0xe7, 0xd6,  
    0x7a, 0x4b, 0x18, 0x29, 0xbe, 0x8f, 0xdc, 0xed, 0xc3, 0xf2, 0xa1, 0x90, 0x07, 0x36, 0x65, 0x54,  
    0x39, 0x08, 0x5b, 0x6a, 0xfd, 0xcc, 0x9f, 0xae, 0x80, 0xb1, 0xe2, 0xd3, 0x44, 0x75, 0x26, 0x17,  
    0xfc, 0xcd, 0x9e, 0xaf, 0x38, 0x09, 0x5a, 0x6b, 0x45, 0x74, 0x27, 0x16, 0x81, 0xb0, 0xe3, 0xd2,  
    0xbf, 0x8e, 0xdd, 0xec, 0x7b, 0x4a, 0x19, 0x28, 0x06, 0x37, 0x64, 0x55, 0xc2, 0xf3, 0xa0, 0x91,  
    0x47, 0x76, 0x25, 0x14, 0x83, 0xb2, 0xe1, 0xd0, 0xfe, 0xcf, 0x9c, 0xad, 0x3a, 0x0b, 0x58, 0x69,  
    0x04, 0x35, 0x66, 0x57, 0xc0, 0xf1, 0xa2, 0x93, 0xbd, 0x8c, 0xdf, 0xee, 0x79, 0x48, 0x1b, 0x2a,  
    0xc1, 0xf0, 0xa3, 0x92, 0x05, 0x34, 0x67, 0x56, 0x78, 0x49, 0x1a, 0x2b, 0xbc, 0x8d, 0xde, 0xef,  
    0x82, 0xb3, 0xe0, 0xd1, 0x46, 0x77, 0x24, 0x15, 0x3b, 0x0a, 0x59, 0x68, 0xff, 0xce, 0x9d, 0xac  
}
```

### 5.7.3.22 data\_async

```
bool data_async = true
```



#### 5.7.3.23 dataformat

```
unsigned int dataformat = BINARY
```

#### 5.7.3.24 diagnostic\_usecs

```
elapsedMicros diagnostic_usecs
```

#### 5.7.3.25 diagnostics

```
bool diagnostics = false
```

#### 5.7.3.26 elapsed\_usecs

```
elapsedMicros elapsed_usecs
```

#### 5.7.3.27 fMPin

```
const int fMPin = CLK_PIN
```

#### 5.7.3.28 frame\_header\_ring

```
TCD1304Device::Frame_Header frame_header_ring[NBUFFERS] = {0}
```

#### 5.7.3.29 frame\_index

```
unsigned int frame_index = 0
```

#### 5.7.3.30 frame\_send\_index

```
unsigned int frame_send_index = 0
```

#### 5.7.3.31 framep

```
TCD1304Device::Frame_Header* framep = &frame_header_ring[0]
```

#### 5.7.3.32 ICGPin

```
const int ICGPin = ICG_PIN
```

**5.7.3.33 nrcvbuf**

```
uint16_t nrcvbuf = 0
```

**5.7.3.34 ocpinattached**

```
bool ocpinattached = false
```

**5.7.3.35 ocpinstate**

```
bool ocpinstate = false
```

**5.7.3.36 rcvbuffer**

```
char rcvbuffer[RCVLEN]
```

**5.7.3.37 SDIPin**

```
const int SDIPin = 11
```

**5.7.3.38 SDOPin**

```
const int SDOPin = 12
```

**5.7.3.39 sensorstr**

```
const char sensorstr[] = "TCD1304"
```

**5.7.3.40 SHPin**

```
const int SHPin = SH_PIN
```

**5.7.3.41 sndbuffer**

```
char sndbuffer[SNDLEN]
```

**5.7.3.42 srcfilestr**

```
const char srcfilestr[] = __FILE__
```

#### 5.7.3.43 sum\_enable

```
bool sum_enable = false
```

#### 5.7.3.44 syncPin

```
const int syncPin = SYNC_PIN
```

#### 5.7.3.45 tcd1304device

```
TCD1304Device tcd1304device = TCD1304Device( )
```

#### 5.7.3.46 triggerPin

```
const int triggerPin = TRIGGER_PIN
```

#### 5.7.3.47 usb\_string\_manufacturer\_name

```
usb_string_descriptor_struct_manufacturer usb_string_manufacturer_name
```

Initial value:

```
= {  
    2 + thisMANUFACTURER_NAME_LEN * 2,  
    3,  
    thisMANUFACTURER_NAME  
}
```

#### 5.7.3.48 usb\_string\_product\_name

```
usb_string_descriptor_struct_product usb_string_product_name
```

Initial value:

```
= {  
    2 + thisPRODUCT_NAME_LEN * 2,  
    3,  
    thisPRODUCT_NAME  
}
```

#### 5.7.3.49 usb\_string\_serial\_number

```
usb_string_descriptor_struct_serial_number usb_string_serial_number
```

Initial value:

```
=  
{  
    2 + thisPRODUCT_SERIAL_NUMBER_LEN * 2,  
    3,  
    thisPRODUCT_SERIAL_NUMBER  
}
```

#### 5.7.3.50 versionstr

```
const char versionstr[] = "TCD1304Device vers 0.4 " __DATE__
```



# Index

- adc
  - TCD1304Device\_Controller\_251208.ino, [174](#)
- adc\_averages
  - TCD1304Device\_Controller\_251208.ino, [174](#)
- adc\_data
  - TCD1304Device\_Controller\_251208.ino, [174](#)
- ADC\_RESOLUTION
  - TCD1304Device\_Controller\_251208.ino, [143](#)
- analogPin
  - TCD1304Device\_Controller\_251208.ino, [174](#)
- ASCII
  - TCD1304Device\_Controller\_251208.ino, [143](#)
- attach\_isr
  - TCD1304Device, [19](#)
- authorstr
  - TCD1304Device\_Controller\_251208.ino, [174](#)
- avgdummy
  - TCD1304Device::Frame\_Header\_struct, [7](#)
- basenamef
  - parselib2.cpp, [68](#)
  - parselib2.h, [79](#)
- bDescriptorType
  - usb\_string\_descriptor\_struct\_manufacturer, [65](#)
  - usb\_string\_descriptor\_struct\_product, [66](#)
  - usb\_string\_descriptor\_struct\_serial\_number, [66](#)
- BINARY
  - TCD1304Device\_Controller\_251208.ino, [143](#)
- bLength
  - usb\_string\_descriptor\_struct\_manufacturer, [65](#)
  - usb\_string\_descriptor\_struct\_product, [66](#)
  - usb\_string\_descriptor\_struct\_serial\_number, [66](#)
- blink
  - TCD1304Device\_Controller\_251208.ino, [146](#)
- buffer
  - TCD1304Device::Frame\_Header\_struct, [7](#)
- buffer\_index
  - TCD1304Device\_Controller\_251208.ino, [174](#)
- buffer\_ring
  - TCD1304Device\_Controller\_251208.ino, [175](#)
- bufferp
  - TCD1304Device\_Controller\_251208.ino, [175](#)
- BUSY\_PIN
  - TCD1304Device2.h, [91](#)
- BUSY\_PIN\_DEFAULT
  - TCD1304Device2.h, [91](#)
- busyPin
  - TCD1304Device\_Controller\_251208.ino, [175](#)
- busytoggled
  - TCD1304Device, [58](#)
- calculateCRC
  - TCD1304Device\_Controller\_251208.ino, [146](#)
- check\_submodule
  - TCD1304Device, [19](#)
- chipTemp\_averages
  - TCD1304Device\_Controller\_251208.ino, [175](#)
- clear\_busypin
  - TCD1304Device, [19](#)
- clear\_error\_flags
  - TCD1304Device, [20](#)
- clear\_frames\_completed\_callback
  - TCD1304Device, [20](#)
- clear\_framesets\_completed\_callback
  - TCD1304Device, [20](#)
- clear\_ldok
  - TCD1304Device, [20](#)
- clear\_mode
  - TCD1304Device, [20](#)
- clear\_run
  - TCD1304Device, [20](#)
- clear\_sync\_busy\_pins
  - TCD1304Device, [21](#)
- clear\_syncpin
  - TCD1304Device, [21](#)
- CLEARCNVST
  - TCD1304Device2.h, [91](#)
- clk
  - TCD1304Device, [58](#)
- CLK\_CHANNEL
  - TCD1304Device2.h, [91](#)
- CLK\_CMPF\_MASK
  - TCD1304Device2.h, [91](#)
- CLK\_CTRL2\_MASK
  - TCD1304Device2.h, [91](#)
- CLK\_DEFAULT
  - TCD1304Device2.h, [92](#)
- CLK\_IRQ
  - TCD1304Device2.h, [92](#)
- CLK\_MASK
  - TCD1304Device2.h, [92](#)
- CLK\_MONITOR\_PIN

TCD1304Device2.h, 92  
 CLK\_MUXVAL  
     TCD1304Device2.h, 92  
 CLK\_PIN  
     TCD1304Device2.h, 92  
 CLK\_SUBMODULE  
     TCD1304Device2.h, 92  
 CLKPin  
     TCD1304Device\_Controller\_251208.ino, 175  
 clock\_callback  
     TCD1304Device\_Controller\_251208.ino, 175  
 clock\_counter  
     TCD1304Device\_Controller\_251208.ino, 175  
 clock\_counts  
     TCD1304Device\_Controller\_251208.ino, 175  
 clock\_isr  
     TCD1304Device\_Controller\_251208.ino, 147  
 clock\_mode  
     TCD1304Device\_Controller\_251208.ino, 175  
 clock\_setup  
     TCD1304Device\_Controller\_251208.ino, 147  
 clock\_start  
     TCD1304Device\_Controller\_251208.ino, 147  
 clock\_stop  
     TCD1304Device\_Controller\_251208.ino, 148  
 clock\_timer  
     TCD1304Device\_Controller\_251208.ino, 175  
 clock\_usecs  
     TCD1304Device\_Controller\_251208.ino, 176  
 close  
     TCD1304Device, 21  
 CMPF\_MASKA\_OFF  
     TCD1304Device2.h, 92  
 CMPF\_MASKA\_ON  
     TCD1304Device2.h, 92  
 CMPF\_MASKA\_ON\_OFF  
     TCD1304Device2.h, 92  
 CMPF\_MASKB\_OFF  
     TCD1304Device2.h, 93  
 CMPF\_MASKB\_ON  
     TCD1304Device2.h, 93  
 CMPF\_MASKB\_ON\_OFF  
     TCD1304Device2.h, 93  
 cnvst  
     TCD1304Device, 58  
 CNVST\_CHANNEL  
     TCD1304Device2.h, 93  
 CNVST\_CMPF\_MASK  
     TCD1304Device2.h, 93  
 cnvst\_counter  
     TCD1304Device, 58  
 CNVST\_CTRL2\_MASK  
     TCD1304Device2.h, 93  
 cnvst\_extra\_delay\_counts  
     TCD1304Device, 58  
 CNVST\_IRQ  
     TCD1304Device2.h, 93  
 CNVST\_MASK  
     TCD1304Device2.h, 93  
 CNVST\_PIN  
     TCD1304Device2.h, 93  
 CNVST\_PULSE\_SECS  
     TCD1304Device2.h, 93  
 CNVST\_SUBMODULE  
     TCD1304Device2.h, 94  
 CNVSTPin  
     TCD1304Device\_Controller\_251208.ino, 176  
 CONTROLLER\_CSPIN  
     TCD1304Device\_Controller\_251208.ino, 144  
 CONTROLLER\_CSPIN2  
     TCD1304Device\_Controller\_251208.ino, 144  
 CONTROLLER\_OCPIN  
     TCD1304Device\_Controller\_251208.ino, 144  
 counter  
     TCD1304Device\_Controller\_251208.ino, 176  
 COUNTER\_MAX\_SECS  
     TCD1304Device2.h, 94  
 countWords  
     parselib2.cpp, 68  
     parselib2.h, 79  
 crc\_enable  
     TCD1304Device\_Controller\_251208.ino, 176  
 crctable  
     TCD1304Device\_Controller\_251208.ino, 176  
 ctrl2\_mask  
     TCD1304Device::SubModule, 11  
 CYCCNT2SECS  
     TCD1304Device2.h, 94  
 cycles64  
     TCD1304Device, 21  
     TCD1304Device\_Controller\_251208.ino, 148  
 CYCLES\_PER\_USEC  
     TCD1304Device\_Controller\_251208.ino, 144  
 data\_async  
     TCD1304Device\_Controller\_251208.ino, 176  
 dataformat  
     TCD1304Device\_Controller\_251208.ino, 176  
 DATASTART  
     TCD1304Device2.h, 94  
 DATASTOP  
     TCD1304Device2.h, 94  
 DEBUGPRINTF  
     TCD1304Device2.h, 94  
 diagnostic\_usecs  
     TCD1304Device\_Controller\_251208.ino, 177  
 diagnostics  
     TCD1304Device\_Controller\_251208.ino, 177

- disable\_irqs
  - TCD1304Device, [22](#)
- disablePinInterrupts
  - TCD1304Device\_Controller\_251208.ino, [149](#)
- divider
  - TCD1304Device::SubModule, [11](#)
- DRMCNELSONLAB
  - TCD1304Device\_Controller\_251208.ino, [144](#)
- eeErase
  - TCD1304Device\_Controller\_251208.ino, [149](#)
- EEPROM\_COEFF\_ADDR
  - TCD1304Device\_Controller\_251208.ino, [144](#)
- EEPROM\_COEFF\_LEN
  - TCD1304Device\_Controller\_251208.ino, [144](#)
- EEPROM\_ID\_ADDR
  - TCD1304Device\_Controller\_251208.ino, [144](#)
- EEPROM\_ID\_LEN
  - TCD1304Device\_Controller\_251208.ino, [144](#)
- EEPROM\_NCOEFFS
  - TCD1304Device\_Controller\_251208.ino, [144](#)
- EEPROM\_NUNITS
  - TCD1304Device\_Controller\_251208.ino, [145](#)
- EEPROM\_SIZE
  - TCD1304Device\_Controller\_251208.ino, [145](#)
- EEPROM\_UNITS\_ADDR
  - TCD1304Device\_Controller\_251208.ino, [145](#)
- EEPROM\_UNITS\_LEN
  - TCD1304Device\_Controller\_251208.ino, [145](#)
- eeread
  - TCD1304Device\_Controller\_251208.ino, [149](#)
- eereadUntil
  - TCD1304Device\_Controller\_251208.ino, [149](#)
- eewrite
  - TCD1304Device\_Controller\_251208.ino, [150](#)
- eewriteUntil
  - TCD1304Device\_Controller\_251208.ino, [150](#)
- elapsed\_usec
  - TCD1304Device\_Controller\_251208.ino, [177](#)
- eos
  - parselib2.cpp, [69](#)
- eow
  - parselib2.cpp, [69](#)
- eraseCoefficients
  - TCD1304Device\_Controller\_251208.ino, [151](#)
- eraseIdentifier
  - TCD1304Device\_Controller\_251208.ino, [151](#)
- eraseUnits
  - TCD1304Device\_Controller\_251208.ino, [152](#)
- error\_flag
  - TCD1304Device, [58](#)
  - TCD1304Device::Frame\_Header\_struct, [8](#)
- fastAnalogRead
  - TCD1304Device\_Controller\_251208.ino, [153](#)
- fill\_frame\_header
  - TCD1304Device, [22](#)
- filler
  - TCD1304Device::SubModule, [11](#)
- Firmware for the TCD1304 using FlexPWM, [1](#)
- flexpwm
  - TCD1304Device, [58](#)
  - TCD1304Device::SubModule, [11](#)
- flexpwm\_running
  - TCD1304Device, [58](#)
- flexpwm\_start
  - TCD1304Device, [23](#)
- flexpwm\_stop
  - TCD1304Device, [23](#)
- flexpwm\_wait
  - TCD1304Device, [24](#)
- fMPin
  - TCD1304Device\_Controller\_251208.ino, [177](#)
- force
  - TCD1304Device, [24](#)
- frame\_callback
  - TCD1304Device\_Controller\_251208.ino, [153](#)
- frame\_counter
  - TCD1304Device, [58](#)
  - TCD1304Device::Frame\_Header\_struct, [8](#)
- frame\_counts
  - TCD1304Device, [58](#)
- frame\_elapsed\_secs
  - TCD1304Device::Frame\_Header\_struct, [8](#)
- frame\_exposure\_secs
  - TCD1304Device::Frame\_Header\_struct, [8](#)
- Frame\_Header
  - TCD1304Device, [18](#)
- frame\_header\_ring
  - TCD1304Device\_Controller\_251208.ino, [177](#)
- frame\_index
  - TCD1304Device\_Controller\_251208.ino, [177](#)
- frame\_send\_index
  - TCD1304Device\_Controller\_251208.ino, [177](#)
- framep
  - TCD1304Device\_Controller\_251208.ino, [177](#)
- frames\_completed
  - TCD1304Device::Frame\_Header\_struct, [8](#)
- frames\_completed\_callback
  - TCD1304Device, [59](#)
- FRAMESET
  - TCD1304Device2.h, [100](#)
- frameset\_arm
  - TCD1304Device, [24](#)
- frameset\_armed
  - TCD1304Device, [59](#)
- frameset\_cnvtst\_isr
  - TCD1304Device, [24](#)
- frameset\_counter

- TCD1304Device, [59](#)
  - TCD1304Device::Frame\_Header\_struct, [8](#)
- frameset\_counts
  - TCD1304Device, [59](#)
- frameset\_icg\_isr
  - TCD1304Device, [25](#)
- frameset\_init\_frames
  - TCD1304Device, [25](#)
- frameset\_init\_frameset
  - TCD1304Device, [26](#)
- frameset\_sh\_isr
  - TCD1304Device, [26](#)
- frameset\_start
  - TCD1304Device, [27](#)
- framesets\_completed
  - TCD1304Device::Frame\_Header\_struct, [8](#)
- framesets\_completed\_callback
  - TCD1304Device, [59](#)
- help
  - TCD1304Device\_Controller\_251208.ino, [153](#)
- icg
  - TCD1304Device, [59](#)
- ICG\_CHANNEL
  - TCD1304Device2.h, [94](#)
- ICG\_CMPF\_MASK
  - TCD1304Device2.h, [94](#)
- icg\_counter
  - TCD1304Device, [59](#)
- ICG\_CTRL2\_MASK
  - TCD1304Device2.h, [94](#)
- ICG\_IRQ
  - TCD1304Device2.h, [95](#)
- ICG\_MASK
  - TCD1304Device2.h, [95](#)
- ICG\_MUXVAL
  - TCD1304Device2.h, [95](#)
- ICG\_PIN
  - TCD1304Device2.h, [95](#)
- ICG\_SUBMODULE
  - TCD1304Device2.h, [95](#)
- ICGPin
  - TCD1304Device\_Controller\_251208.ino, [177](#)
- IMR\_INDEX
  - TCD1304Device\_Controller\_251208.ino, [145](#)
- inten\_mask
  - TCD1304Device::SubModule, [11](#)
- intena\_mask
  - TCD1304Device::SubModule, [11](#)
- invertA
  - TCD1304Device::SubModule, [11](#)
- invertB
  - TCD1304Device::SubModule, [11](#)
- irq
  - TCD1304Device::SubModule, [11](#)
- isr
  - TCD1304Device::SubModule, [11](#)
- ISR\_INDEX
  - TCD1304Device\_Controller\_251208.ino, [145](#)
- load\_frames\_completed\_callback
  - TCD1304Device, [28](#)
- load\_framesets\_completed\_callback
  - TCD1304Device, [28](#)
- load\_submodule
  - TCD1304Device, [28](#)
- loop
  - TCD1304Device\_Controller\_251208.ino, [154](#)
- mask
  - TCD1304Device::SubModule, [12](#)
- mode
  - TCD1304Device, [59](#)
  - TCD1304Device::Frame\_Header\_struct, [8](#)
- muxvalA
  - TCD1304Device::SubModule, [12](#)
- muxvalB
  - TCD1304Device::SubModule, [12](#)
- NADC\_CHANNELS
  - TCD1304Device\_Controller\_251208.ino, [145](#)
- name
  - TCD1304Device::SubModule, [12](#)
- NBITS
  - TCD1304Device2.h, [95](#)
- nbuffer
  - TCD1304Device::Frame\_Header\_struct, [8](#)
- NBUFFERS
  - TCD1304Device\_Controller\_251208.ino, [145](#)
- NBYTES
  - TCD1304Device2.h, [95](#)
- NBYTES32
  - TCD1304Device2.h, [95](#)
- NDARK
  - TCD1304Device2.h, [95](#)
- newvals
  - TCD1304Device::SubModule, [12](#)
- nextWord
  - parselib2.cpp, [69](#)
  - parselib2.h, [79](#)
- NOTCONFIGURED
  - TCD1304Device2.h, [100](#)
- NPIXELS
  - TCD1304Device2.h, [95](#)
- nprintbuffer
  - parselib2.cpp, [77](#)
- nrcvbuf
  - TCD1304Device\_Controller\_251208.ino, [177](#)
- NREADOUT



- TCD1304Device2.h, 96
- ocpinAttach
  - TCD1304Device\_Controller\_251208.ino, 155
- ocpinattached
  - TCD1304Device\_Controller\_251208.ino, 178
- ocpinDetach
  - TCD1304Device\_Controller\_251208.ino, 156
- ocpinISR
  - TCD1304Device\_Controller\_251208.ino, 156
- ocpinRead
  - TCD1304Device\_Controller\_251208.ino, 157
- ocpinstate
  - TCD1304Device\_Controller\_251208.ino, 178
- offA\_counts
  - TCD1304Device::SubModule, 12
- offB\_counts
  - TCD1304Device::SubModule, 12
- offset
  - TCD1304Device::Frame\_Header\_struct, 8
- onA\_counts
  - TCD1304Device::SubModule, 12
- onB\_counts
  - TCD1304Device::SubModule, 12
- oops\_flag
  - TCD1304Device, 59
  - TCD1304Device::Frame\_Header\_struct, 9
- parselib2.cpp, 67
  - basenamef, 68
  - countWords, 68
  - eos, 69
  - eow, 69
  - nextWord, 69
  - nprintbuffer, 77
  - printbuffer, 77
  - scaling, 70
  - serialPrintf, 70
  - serialPrintInf, 71
  - strBool, 71
  - strFlt, 72
  - strFlts, 72
  - strMatch, 73
  - strUInt, 74
  - strUInt16, 74
  - strUInt32, 75
  - strUInt32s, 75
  - strUInt8, 76
  - strUInt8lim, 76
  - strUInts, 77
  - wordLength, 77
- parselib2.h, 78
  - basenamef, 79
  - countWords, 79
  - nextWord, 79
  - PARSELIB\_H, 79
  - serialPrintf, 80
  - serialPrintInf, 80
  - strBool, 81
  - strFlt, 81
  - strFlts, 82
  - strMatch, 83
  - strUInt, 83
  - strUInt16, 84
  - strUInt32, 84
  - strUInt32s, 85
  - strUInt8, 85
  - strUInt8lim, 86
  - strUInts, 86
  - wordLength, 87
- PARSELIB\_H
  - parselib2.h, 79
- period\_counts
  - TCD1304Device::SubModule, 12
- period\_secs
  - TCD1304Device::SubModule, 13
- pinA
  - TCD1304Device::SubModule, 13
- pinB
  - TCD1304Device::SubModule, 13
- PINPULLS
  - TCD1304Device2.h, 96
- prescale
  - TCD1304Device::SubModule, 13
- print\_and\_check\_submodule
  - TCD1304Device, 28
- print\_counters
  - TCD1304Device, 29
- print\_errormsg
  - TCD1304Device, 29, 30
- print\_submodule
  - TCD1304Device, 30
- printbits16\_
  - TCD1304Device, 31
- printbuffer
  - parselib2.cpp, 77
- printCoefficients
  - TCD1304Device\_Controller\_251208.ino, 157
- printIdentifier
  - TCD1304Device\_Controller\_251208.ino, 158
- printTriggerSetup
  - TCD1304Device\_Controller\_251208.ino, 158
- printUnits
  - TCD1304Device\_Controller\_251208.ino, 159
- PULSE
  - TCD1304Device2.h, 100
- pulse\_arm
  - TCD1304Device, 31
- pulse\_armed

- TCD1304Device, 60
- pulse\_cnvst\_isr
  - TCD1304Device, 31
- pulse\_icg\_isr
  - TCD1304Device, 31
- pulse\_init\_frames
  - TCD1304Device, 32
- pulse\_init\_frameset
  - TCD1304Device, 32
- pulse\_sh\_isr
  - TCD1304Device, 33
- pulse\_start
  - TCD1304Device, 33
- pulsePin
  - TCD1304Device\_Controller\_251208.ino, 159
- PWM\_CTRL2\_CLOCK\_MASTER
  - TCD1304Device2.h, 96
- PWM\_CTRL2\_CLOCK\_SLAVE
  - TCD1304Device2.h, 96
- PWM\_CTRL2\_CLOCK\_SYNC
  - TCD1304Device2.h, 96
- rcvbuffer
  - TCD1304Device\_Controller\_251208.ino, 178
- RCVLEN
  - TCD1304Device\_Controller\_251208.ino, 145
- read
  - TCD1304Device, 34, 35
- read\_buffer
  - TCD1304Device, 60
- read\_callback
  - TCD1304Device, 60
- read\_counter
  - TCD1304Device, 60
- read\_counts
  - TCD1304Device, 60
- read\_expected\_time
  - TCD1304Device, 60
- read\_pointer
  - TCD1304Device, 60
- readCoefficients
  - TCD1304Device\_Controller\_251208.ino, 160
- readIdentifier
  - TCD1304Device\_Controller\_251208.ino, 160
- README.md, 88
- readUnits
  - TCD1304Device\_Controller\_251208.ino, 161
- ready\_for\_send
  - TCD1304Device::Frame\_Header\_struct, 9
- register\_dump
  - TCD1304Device, 36
- resumePinInterrupts
  - TCD1304Device\_Controller\_251208.ino, 162
- ROUNDTO
  - TCD1304Device2.h, 96
- ROUNDTOMOD
  - TCD1304Device2.h, 96
- ROUNDUP
  - TCD1304Device2.h, 96
- scaling
  - parselib2.cpp, 70
- SDIPin
  - TCD1304Device\_Controller\_251208.ino, 178
- SDOPin
  - TCD1304Device\_Controller\_251208.ino, 178
- send\_frame
  - TCD1304Device\_Controller\_251208.ino, 162
- send\_frames
  - TCD1304Device\_Controller\_251208.ino, 162
- send\_header
  - TCD1304Device\_Controller\_251208.ino, 163
- sendADCs
  - TCD1304Device\_Controller\_251208.ino, 163
- sendBuffer\_Binary
  - TCD1304Device\_Controller\_251208.ino, 164
- sendBuffer\_Formatted
  - TCD1304Device\_Controller\_251208.ino, 164
- sendChipTemperature
  - TCD1304Device\_Controller\_251208.ino, 165
- sendData
  - TCD1304Device\_Controller\_251208.ino, 165
- sendDataCRC
  - TCD1304Device\_Controller\_251208.ino, 166
- sendDataReady
  - TCD1304Device\_Controller\_251208.ino, 166
- sendDataSum
  - TCD1304Device\_Controller\_251208.ino, 166
- sendTestData
  - TCD1304Device\_Controller\_251208.ino, 167
- sensorstr
  - TCD1304Device\_Controller\_251208.ino, 178
- serialPrintf
  - parselib2.cpp, 70
  - parselib2.h, 80
- serialPrintInf
  - parselib2.cpp, 71
  - parselib2.h, 80
- set\_clock\_master
  - TCD1304Device, 36, 37
- set\_clock\_slave
  - TCD1304Device, 37, 38
- set\_clock\_sync
  - TCD1304Device, 38
- set\_init
  - TCD1304Device, 39
- set\_Idok
  - TCD1304Device, 39

- set\_outen
  - TCD1304Device, [39](#)
- set\_outen\_off
  - TCD1304Device, [39](#)
- set\_outen\_on
  - TCD1304Device, [39](#)
- set\_outenA\_on
  - TCD1304Device, [40](#)
- set\_prescale
  - TCD1304Device, [40](#)
- set\_run
  - TCD1304Device, [40](#)
- set\_val0
  - TCD1304Device, [40](#)
- set\_val1
  - TCD1304Device, [40](#)
- set\_val2
  - TCD1304Device, [40](#)
- set\_val3
  - TCD1304Device, [40](#)
- set\_val4
  - TCD1304Device, [41](#)
- set\_val5
  - TCD1304Device, [41](#)
- SETCNVST
  - TCD1304Device2.h, [97](#)
- setup
  - TCD1304Device\_Controller\_251208.ino, [168](#)
- setup\_digital\_pins
  - TCD1304Device, [41](#)
- setup\_frameset
  - TCD1304Device, [41](#)
- setup\_pulse
  - TCD1304Device, [42](#)
- setup\_submodule
  - TCD1304Device, [43](#)
- setup\_timer
  - TCD1304Device, [44](#)
- setup\_triggers
  - TCD1304Device, [45](#)
- sh
  - TCD1304Device, [60](#)
- SH\_CHANNEL
  - TCD1304Device2.h, [97](#)
- sh\_clearing\_counter
  - TCD1304Device, [60](#)
- sh\_clearing\_counts
  - TCD1304Device, [61](#)
- SH\_CLEARING\_DEFAULT
  - TCD1304Device2.h, [97](#)
- SH\_CMPF\_MASK
  - TCD1304Device2.h, [97](#)
- sh\_counter
  - TCD1304Device, [61](#)
- sh\_counts\_per\_icg
  - TCD1304Device, [61](#)
- SH\_CTRL2\_MASK
  - TCD1304Device2.h, [97](#)
- sh\_cyccnt64\_exposure
  - TCD1304Device, [61](#)
- sh\_cyccnt64\_now
  - TCD1304Device, [61](#)
- sh\_cyccnt64\_prev
  - TCD1304Device, [61](#)
- sh\_cyccnt64\_start
  - TCD1304Device, [61](#)
- sh\_difference\_secs
  - TCD1304Device, [45](#)
- sh\_elapsed\_secs
  - TCD1304Device, [46](#)
- sh\_exposure\_secs
  - TCD1304Device, [46](#)
- SH\_IRQ
  - TCD1304Device2.h, [97](#)
- SH\_MASK
  - TCD1304Device2.h, [97](#)
- SH\_MUXVAL
  - TCD1304Device2.h, [97](#)
- SH\_PIN
  - TCD1304Device2.h, [98](#)
- sh\_short\_period\_counts
  - TCD1304Device, [61](#)
- SH\_STOP\_IN\_READ
  - TCD1304Device2.h, [98](#)
- SH\_SUBMODULE
  - TCD1304Device2.h, [98](#)
- SHPin
  - TCD1304Device\_Controller\_251208.ino, [178](#)
- SHUTTERMIN
  - TCD1304Device2.h, [98](#)
- skip\_one
  - TCD1304Device, [61](#)
- skip\_one\_reload
  - TCD1304Device, [61](#)
- sndbuffer
  - TCD1304Device\_Controller\_251208.ino, [178](#)
- SNDLEN
  - TCD1304Device\_Controller\_251208.ino, [145](#)
- spi\_settings
  - TCD1304Device\_Controller\_251208.ino, [168](#)
- srcfilestr
  - TCD1304Device\_Controller\_251208.ino, [178](#)
- start\_read
  - TCD1304Device, [46](#)
- start\_triggers
  - TCD1304Device, [47](#)
- stop\_all
  - TCD1304Device, [47](#)

- stop\_runs\_only
  - TCD1304Device, [48](#)
- stop\_triggers
  - TCD1304Device, [48](#)
- stop\_with\_irqs
  - TCD1304Device, [49](#)
- storeCoefficients
  - TCD1304Device\_Controller\_251208.ino, [168](#)
- storeIdentifier
  - TCD1304Device\_Controller\_251208.ino, [169](#)
- storeUnits
  - TCD1304Device\_Controller\_251208.ino, [170](#)
- strBool
  - parselib2.cpp, [71](#)
  - parselib2.h, [81](#)
- strFlt
  - parselib2.cpp, [72](#)
  - parselib2.h, [81](#)
- strFlts
  - parselib2.cpp, [72](#)
  - parselib2.h, [82](#)
- strMatch
  - parselib2.cpp, [73](#)
  - parselib2.h, [83](#)
- strPin
  - TCD1304Device\_Controller\_251208.ino, [170](#)
- strSubModule
  - TCD1304Device\_Controller\_251208.ino, [171](#)
- strTrigger
  - TCD1304Device\_Controller\_251208.ino, [172](#)
- strUInt
  - parselib2.cpp, [74](#)
  - parselib2.h, [83](#)
- strUInt16
  - parselib2.cpp, [74](#)
  - parselib2.h, [84](#)
- strUInt32
  - parselib2.cpp, [75](#)
  - parselib2.h, [84](#)
- strUInt32s
  - parselib2.cpp, [75](#)
  - parselib2.h, [85](#)
- strUInt8
  - parselib2.cpp, [76](#)
  - parselib2.h, [85](#)
- strUInt8lim
  - parselib2.cpp, [76](#)
  - parselib2.h, [86](#)
- strUints
  - parselib2.cpp, [77](#)
  - parselib2.h, [86](#)
- submod
  - TCD1304Device::SubModule, [13](#)
- SubModule
  - TCD1304Device::SubModule, [10](#)
- sum\_enable
  - TCD1304Device\_Controller\_251208.ino, [178](#)
- sumData
  - TCD1304Device\_Controller\_251208.ino, [173](#)
- sync\_enabled
  - TCD1304Device, [62](#)
- SYNC\_PIN
  - TCD1304Device2.h, [98](#)
- sync\_pin
  - TCD1304Device, [62](#)
- SYNC\_PIN\_DEFAULT
  - TCD1304Device2.h, [98](#)
- syncPin
  - TCD1304Device\_Controller\_251208.ino, [179](#)
- sync toggled
  - TCD1304Device, [62](#)
- TCD1304\_MAXCLKHZ
  - TCD1304Device2.h, [98](#)
- TCD1304\_MINCLKHZ
  - TCD1304Device2.h, [98](#)
- TCD1304\_Mode\_t
  - TCD1304Device2.h, [100](#)
- TCD1304Device, [14](#)
  - attach\_isr, [19](#)
  - busy toggled, [58](#)
  - check\_submodule, [19](#)
  - clear\_busypin, [19](#)
  - clear\_error\_flags, [20](#)
  - clear\_frames\_completed\_callback, [20](#)
  - clear\_framesets\_completed\_callback, [20](#)
  - clear\_ldok, [20](#)
  - clear\_mode, [20](#)
  - clear\_run, [20](#)
  - clear\_sync\_busy\_pins, [21](#)
  - clear\_syncpin, [21](#)
  - clk, [58](#)
  - close, [21](#)
  - cnvst, [58](#)
  - cnvst\_counter, [58](#)
  - cnvst\_extra\_delay\_counts, [58](#)
  - cycles64, [21](#)
  - disable\_irqs, [22](#)
  - error\_flag, [58](#)
  - fill\_frame\_header, [22](#)
  - flexpwm, [58](#)
  - flexpwm\_running, [58](#)
  - flexpwm\_start, [23](#)
  - flexpwm\_stop, [23](#)
  - flexpwm\_wait, [24](#)
  - force, [24](#)
  - frame\_counter, [58](#)
  - frame\_counts, [58](#)

Frame\_Header, 18  
frames\_completed\_callback, 59  
frameset\_arm, 24  
frameset\_armed, 59  
frameset\_cnvst\_isr, 24  
frameset\_counter, 59  
frameset\_counts, 59  
frameset\_icg\_isr, 25  
frameset\_init\_frames, 25  
frameset\_init\_frameset, 26  
frameset\_sh\_isr, 26  
frameset\_start, 27  
framesets\_completed\_callback, 59  
icg, 59  
icg\_counter, 59  
load\_frames\_completed\_callback, 28  
load\_framesets\_completed\_callback, 28  
load\_submodule, 28  
mode, 59  
oops\_flag, 59  
print\_and\_check\_submodule, 28  
print\_counters, 29  
print\_errormsg, 29, 30  
print\_submodule, 30  
printbits16\_, 31  
pulse\_arm, 31  
pulse\_armed, 60  
pulse\_cnvst\_isr, 31  
pulse\_icg\_isr, 31  
pulse\_init\_frames, 32  
pulse\_init\_frameset, 32  
pulse\_sh\_isr, 33  
pulse\_start, 33  
read, 34, 35  
read\_buffer, 60  
read\_callback, 60  
read\_counter, 60  
read\_counts, 60  
read\_expected\_time, 60  
read\_pointer, 60  
register\_dump, 36  
set\_clock\_master, 36, 37  
set\_clock\_slave, 37, 38  
set\_clock\_sync, 38  
set\_init, 39  
set\_ldok, 39  
set\_outen, 39  
set\_outen\_off, 39  
set\_outen\_on, 39  
set\_outenA\_on, 40  
set\_prescale, 40  
set\_run, 40  
set\_val0, 40  
set\_val1, 40  
set\_val2, 40  
set\_val3, 40  
set\_val4, 41  
set\_val5, 41  
setup\_digital\_pins, 41  
setup\_frameset, 41  
setup\_pulse, 42  
setup\_submodule, 43  
setup\_timer, 44  
setup\_triggers, 45  
sh, 60  
sh\_clearing\_counter, 60  
sh\_clearing\_counts, 61  
sh\_counter, 61  
sh\_counts\_per\_icg, 61  
sh\_cycCNT64\_exposure, 61  
sh\_cycCNT64\_now, 61  
sh\_cycCNT64\_prev, 61  
sh\_cycCNT64\_start, 61  
sh\_difference\_secs, 45  
sh\_elapsed\_secs, 46  
sh\_exposure\_secs, 46  
sh\_short\_period\_counts, 61  
skip\_one, 61  
skip\_one\_reload, 61  
start\_read, 46  
start\_triggers, 47  
stop\_all, 47  
stop\_runs\_only, 48  
stop\_triggers, 48  
stop\_with\_irqs, 49  
sync\_enabled, 62  
sync\_pin, 62  
synctoggled, 62  
TCD1304Device, 18  
timer, 62  
timer\_callback, 62  
timer\_cycCNT64\_now, 62  
timer\_cycCNT64\_prev, 62  
timer\_cycCNT64\_start, 62  
timer\_difference\_secs, 49  
timer\_elapsed\_secs, 50  
timer\_first\_time\_flag, 62  
timer\_inner\_counter, 63  
timer\_inner\_counts, 63  
timer\_interframe\_min\_secs, 63  
timer\_interval\_secs, 63  
timer\_isr, 50  
timer\_outer\_counter, 63  
timer\_outer\_counts, 63  
timer\_period\_secs, 63  
timer\_running, 63  
timer\_start, 51  
timer\_stop, 52

- timer\_stop\_with\_irq, [52](#)
- timer\_wait, [52](#)
- timerflexpwm, [63](#)
- toggle\_busypin, [53](#)
- toggle\_syncpin, [53](#)
- trigger\_attached, [63](#)
- trigger\_busy, [64](#)
- trigger\_callback, [64](#)
- trigger\_counter, [64](#)
- trigger\_counts, [64](#)
- trigger\_cyccnt64\_now, [64](#)
- trigger\_cyccnt64\_prev, [64](#)
- trigger\_cyccnt64\_start, [64](#)
- trigger\_difference\_secs, [53](#)
- trigger\_edge\_mode, [64](#)
- trigger\_elapsed\_secs, [53](#)
- trigger\_isr, [53](#)
- trigger\_mode, [64](#)
- trigger\_pin, [64](#)
- trigger\_pin\_mode, [65](#)
- triggered\_read, [54](#), [55](#)
- update\_read\_buffer, [56](#)
- wait\_read, [56](#)
- wait\_triggered\_read, [57](#)
- wait\_triggers, [57](#)
- tcd1304device
  - TCD1304Device\_Controller\_251208.ino, [179](#)
- TCD1304Device2.h, [88](#)
  - BUSY\_PIN, [91](#)
  - BUSY\_PIN\_DEFAULT, [91](#)
  - CLEARCNVST, [91](#)
  - CLK\_CHANNEL, [91](#)
  - CLK\_CMPF\_MASK, [91](#)
  - CLK\_CTRL2\_MASK, [91](#)
  - CLK\_DEFAULT, [92](#)
  - CLK\_IRQ, [92](#)
  - CLK\_MASK, [92](#)
  - CLK\_MONITOR\_PIN, [92](#)
  - CLK\_MUXVAL, [92](#)
  - CLK\_PIN, [92](#)
  - CLK\_SUBMODULE, [92](#)
  - CMPF\_MASKA\_OFF, [92](#)
  - CMPF\_MASKA\_ON, [92](#)
  - CMPF\_MASKA\_ON\_OFF, [92](#)
  - CMPF\_MASKB\_OFF, [93](#)
  - CMPF\_MASKB\_ON, [93](#)
  - CMPF\_MASKB\_ON\_OFF, [93](#)
  - CNVST\_CHANNEL, [93](#)
  - CNVST\_CMPF\_MASK, [93](#)
  - CNVST\_CTRL2\_MASK, [93](#)
  - CNVST\_IRQ, [93](#)
  - CNVST\_MASK, [93](#)
  - CNVST\_PIN, [93](#)
  - CNVST\_PULSE\_SECS, [93](#)
  - CNVST\_SUBMODULE, [94](#)
  - COUNTER\_MAX\_SECS, [94](#)
  - CYCCNT2SECS, [94](#)
  - DATASTART, [94](#)
  - DATASTOP, [94](#)
  - DEBUGPRINTF, [94](#)
  - FRAMESET, [100](#)
  - ICG\_CHANNEL, [94](#)
  - ICG\_CMPF\_MASK, [94](#)
  - ICG\_CTRL2\_MASK, [94](#)
  - ICG\_IRQ, [95](#)
  - ICG\_MASK, [95](#)
  - ICG\_MUXVAL, [95](#)
  - ICG\_PIN, [95](#)
  - ICG\_SUBMODULE, [95](#)
  - NBITS, [95](#)
  - NBYTES, [95](#)
  - NBYTES32, [95](#)
  - NDARK, [95](#)
  - NOTCONFIGURED, [100](#)
  - NPIXELS, [95](#)
  - NREADOUT, [96](#)
  - PINPULLS, [96](#)
  - PULSE, [100](#)
  - PWM\_CTRL2\_CLOCK\_MASTER, [96](#)
  - PWM\_CTRL2\_CLOCK\_SLAVE, [96](#)
  - PWM\_CTRL2\_CLOCK\_SYNC, [96](#)
  - ROUNDTO, [96](#)
  - ROUNDTOMOD, [96](#)
  - ROUNDUP, [96](#)
  - SETCNVST, [97](#)
  - SH\_CHANNEL, [97](#)
  - SH\_CLEARING\_DEFAULT, [97](#)
  - SH\_CMPF\_MASK, [97](#)
  - SH\_CTRL2\_MASK, [97](#)
  - SH\_IRQ, [97](#)
  - SH\_MASK, [97](#)
  - SH\_MUXVAL, [97](#)
  - SH\_PIN, [98](#)
  - SH\_STOP\_IN\_READ, [98](#)
  - SH\_SUBMODULE, [98](#)
  - SHUTTERMIN, [98](#)
  - SYNC\_PIN, [98](#)
  - SYNC\_PIN\_DEFAULT, [98](#)
  - TCD1304\_MAXCLKHZ, [98](#)
  - TCD1304\_MINCLKHZ, [98](#)
  - TCD1304\_Mode\_t, [100](#)
  - TDIFF, [98](#)
  - TIMER, [100](#)
  - TIMER\_CHANNEL, [99](#)
  - TIMER\_CMPF\_MASK, [99](#)
  - TIMER\_CTRL2\_MASK, [99](#)
  - TIMER\_IRQ, [99](#)
  - TIMER\_MASK, [99](#)

- TIMER\_MUXVAL, [99](#)
- TIMER\_PIN, [99](#)
- TIMER\_SUBMODULE, [99](#)
- TRIGGER\_PIN, [99](#)
- USBSPEED, [100](#)
- USBTRANSFERSECS, [100](#)
- VFS, [100](#)
- VPERBIT, [100](#)
- TCD1304Device::Frame\_Header\_struct, [7](#)
  - avgdummy, [7](#)
  - buffer, [7](#)
  - error\_flag, [8](#)
  - frame\_counter, [8](#)
  - frame\_elapsed\_secs, [8](#)
  - frame\_exposure\_secs, [8](#)
  - frames\_completed, [8](#)
  - frameset\_counter, [8](#)
  - framesets\_completed, [8](#)
  - mode, [8](#)
  - nbuffer, [8](#)
  - offset, [8](#)
  - oops\_flag, [9](#)
  - ready\_for\_send, [9](#)
  - timer\_difference\_secs, [9](#)
  - timer\_elapsed\_secs, [9](#)
  - trigger\_counter, [9](#)
  - trigger\_difference\_secs, [9](#)
  - trigger\_elapsed\_secs, [9](#)
  - trigger\_mode, [9](#)
- TCD1304Device::SubModule, [10](#)
  - ctrl2\_mask, [11](#)
  - divider, [11](#)
  - filler, [11](#)
  - flexpwm, [11](#)
  - inten\_mask, [11](#)
  - intena\_mask, [11](#)
  - invertA, [11](#)
  - invertB, [11](#)
  - irq, [11](#)
  - isr, [11](#)
  - mask, [12](#)
  - muxvalA, [12](#)
  - muxvalB, [12](#)
  - name, [12](#)
  - newvals, [12](#)
  - offA\_counts, [12](#)
  - offB\_counts, [12](#)
  - onA\_counts, [12](#)
  - onB\_counts, [12](#)
  - period\_counts, [12](#)
  - period\_secs, [13](#)
  - pinA, [13](#)
  - pinB, [13](#)
  - prescale, [13](#)
  - submod, [13](#)
  - SubModule, [10](#)
- TCD1304Device\_Controller\_251208.ino, [140](#)
  - adc, [174](#)
  - adc\_averages, [174](#)
  - adc\_data, [174](#)
  - ADC\_RESOLUTION, [143](#)
  - analogPin, [174](#)
  - ASCII, [143](#)
  - authorstr, [174](#)
  - BINARY, [143](#)
  - blink, [146](#)
  - buffer\_index, [174](#)
  - buffer\_ring, [175](#)
  - bufferp, [175](#)
  - busyPin, [175](#)
  - calculateCRC, [146](#)
  - chipTemp\_averages, [175](#)
  - CLKPin, [175](#)
  - clock\_callback, [175](#)
  - clock\_counter, [175](#)
  - clock\_counts, [175](#)
  - clock\_isr, [147](#)
  - clock\_mode, [175](#)
  - clock\_setup, [147](#)
  - clock\_start, [147](#)
  - clock\_stop, [148](#)
  - clock\_timer, [175](#)
  - clock\_usecs, [176](#)
  - CNVSTPin, [176](#)
  - CONTROLLER\_CSPIN, [144](#)
  - CONTROLLER\_CSPIN2, [144](#)
  - CONTROLLER\_OCPIN, [144](#)
  - counter, [176](#)
  - crc\_enable, [176](#)
  - crctable, [176](#)
  - cycles64, [148](#)
  - CYCLES\_PER\_USEC, [144](#)
  - data\_async, [176](#)
  - dataformat, [176](#)
  - diagnostic\_usecs, [177](#)
  - diagnostics, [177](#)
  - disablePinInterrupts, [149](#)
  - DRMCNELSONLAB, [144](#)
  - eeErase, [149](#)
  - EEPROM\_COEFF\_ADDR, [144](#)
  - EEPROM\_COEFF\_LEN, [144](#)
  - EEPROM\_ID\_ADDR, [144](#)
  - EEPROM\_ID\_LEN, [144](#)
  - EEPROM\_NCOEFFS, [144](#)
  - EEPROM\_NUNITS, [145](#)
  - EEPROM\_SIZE, [145](#)
  - EEPROM\_UNITS\_ADDR, [145](#)
  - EEPROM\_UNITS\_LEN, [145](#)

- eeread, [149](#)
- eereadUntil, [149](#)
- eewrite, [150](#)
- eewriteUntil, [150](#)
- elapsed\_usecs, [177](#)
- eraseCoefficients, [151](#)
- eraseIdentifier, [151](#)
- eraseUnits, [152](#)
- fastAnalogRead, [153](#)
- fMPin, [177](#)
- frame\_callback, [153](#)
- frame\_header\_ring, [177](#)
- frame\_index, [177](#)
- frame\_send\_index, [177](#)
- framep, [177](#)
- help, [153](#)
- ICGPIn, [177](#)
- IMR\_INDEX, [145](#)
- ISR\_INDEX, [145](#)
- loop, [154](#)
- NADC\_CHANNELS, [145](#)
- NBUFFERS, [145](#)
- nrcvbuf, [177](#)
- ocpinAttach, [155](#)
- ocpinattached, [178](#)
- ocpinDetach, [156](#)
- ocpinISR, [156](#)
- ocpinRead, [157](#)
- ocpinstate, [178](#)
- printCoefficients, [157](#)
- printIdentifier, [158](#)
- printTriggerSetup, [158](#)
- printUnits, [159](#)
- pulsePin, [159](#)
- rcvbuffer, [178](#)
- RCVLEN, [145](#)
- readCoefficients, [160](#)
- readIdentifier, [160](#)
- readUnits, [161](#)
- resumePinInterrupts, [162](#)
- SDIPin, [178](#)
- SDOPin, [178](#)
- send\_frame, [162](#)
- send\_frames, [162](#)
- send\_header, [163](#)
- sendADCs, [163](#)
- sendBuffer\_Binary, [164](#)
- sendBuffer\_Formatted, [164](#)
- sendChipTemperature, [165](#)
- sendData, [165](#)
- sendDataCRC, [166](#)
- sendDataReady, [166](#)
- sendDataSum, [166](#)
- sendTestData, [167](#)
- sensorstr, [178](#)
- setup, [168](#)
- SHPin, [178](#)
- sndbuffer, [178](#)
- SNDLEN, [145](#)
- spi\_settings, [168](#)
- srcfilestr, [178](#)
- storeCoefficients, [168](#)
- storeIdentifier, [169](#)
- storeUnits, [170](#)
- strPin, [170](#)
- strSubModule, [171](#)
- strTrigger, [172](#)
- sum\_enable, [178](#)
- sumData, [173](#)
- syncPin, [179](#)
- tcd1304device, [179](#)
- tempmonGetTemp, [173](#)
- thisMANUFACTURER\_NAME, [146](#)
- thisMANUFACTURER\_NAME\_LEN, [146](#)
- thisPRODUCT\_NAME, [146](#)
- thisPRODUCT\_NAME\_LEN, [146](#)
- thisPRODUCT\_SERIAL\_NUMBER, [146](#)
- thisPRODUCT\_SERIAL\_NUMBER\_LEN, [146](#)
- triggerPin, [179](#)
- usb\_string\_manufacturer\_name, [179](#)
- usb\_string\_product\_name, [179](#)
- usb\_string\_serial\_number, [179](#)
- versionstr, [179](#)
- TDIFF
  - TCD1304Device2.h, [98](#)
- tempmonGetTemp
  - TCD1304Device\_Controller\_251208.ino, [173](#)
- thisMANUFACTURER\_NAME
  - TCD1304Device\_Controller\_251208.ino, [146](#)
- thisMANUFACTURER\_NAME\_LEN
  - TCD1304Device\_Controller\_251208.ino, [146](#)
- thisPRODUCT\_NAME
  - TCD1304Device\_Controller\_251208.ino, [146](#)
- thisPRODUCT\_NAME\_LEN
  - TCD1304Device\_Controller\_251208.ino, [146](#)
- thisPRODUCT\_SERIAL\_NUMBER
  - TCD1304Device\_Controller\_251208.ino, [146](#)
- thisPRODUCT\_SERIAL\_NUMBER\_LEN
  - TCD1304Device\_Controller\_251208.ino, [146](#)
- TIMER
  - TCD1304Device2.h, [100](#)
- timer
  - TCD1304Device, [62](#)
- timer\_callback
  - TCD1304Device, [62](#)
- TIMER\_CHANNEL
  - TCD1304Device2.h, [99](#)
- TIMER\_CMPF\_MASK



TCD1304Device2.h, 99  
 TIMER\_CTRL2\_MASK  
     TCD1304Device2.h, 99  
 timer\_cyccnt64\_now  
     TCD1304Device, 62  
 timer\_cyccnt64\_prev  
     TCD1304Device, 62  
 timer\_cyccnt64\_start  
     TCD1304Device, 62  
 timer\_difference\_secs  
     TCD1304Device, 49  
     TCD1304Device::Frame\_Header\_struct, 9  
 timer\_elapsed\_secs  
     TCD1304Device, 50  
     TCD1304Device::Frame\_Header\_struct, 9  
 timer\_first\_time\_flag  
     TCD1304Device, 62  
 timer\_inner\_counter  
     TCD1304Device, 63  
 timer\_inner\_counts  
     TCD1304Device, 63  
 timer\_interframe\_min\_secs  
     TCD1304Device, 63  
 timer\_interval\_secs  
     TCD1304Device, 63  
 TIMER\_IRQ  
     TCD1304Device2.h, 99  
 timer\_isr  
     TCD1304Device, 50  
 TIMER\_MASK  
     TCD1304Device2.h, 99  
 TIMER\_MUXVAL  
     TCD1304Device2.h, 99  
 timer\_outer\_counter  
     TCD1304Device, 63  
 timer\_outer\_counts  
     TCD1304Device, 63  
 timer\_period\_secs  
     TCD1304Device, 63  
 TIMER\_PIN  
     TCD1304Device2.h, 99  
 timer\_running  
     TCD1304Device, 63  
 timer\_start  
     TCD1304Device, 51  
 timer\_stop  
     TCD1304Device, 52  
 timer\_stop\_with\_irq  
     TCD1304Device, 52  
 TIMER\_SUBMODULE  
     TCD1304Device2.h, 99  
 timer\_wait  
     TCD1304Device, 52  
 timerflexpwm  
     TCD1304Device, 63  
 toggle\_busypin  
     TCD1304Device, 53  
 toggle\_syncpin  
     TCD1304Device, 53  
 trigger\_attached  
     TCD1304Device, 63  
 trigger\_busy  
     TCD1304Device, 64  
 trigger\_callback  
     TCD1304Device, 64  
 trigger\_counter  
     TCD1304Device, 64  
     TCD1304Device::Frame\_Header\_struct, 9  
 trigger\_counts  
     TCD1304Device, 64  
 trigger\_cyccnt64\_now  
     TCD1304Device, 64  
 trigger\_cyccnt64\_prev  
     TCD1304Device, 64  
 trigger\_cyccnt64\_start  
     TCD1304Device, 64  
 trigger\_difference\_secs  
     TCD1304Device, 53  
     TCD1304Device::Frame\_Header\_struct, 9  
 trigger\_edge\_mode  
     TCD1304Device, 64  
 trigger\_elapsed\_secs  
     TCD1304Device, 53  
     TCD1304Device::Frame\_Header\_struct, 9  
 trigger\_isr  
     TCD1304Device, 53  
 trigger\_mode  
     TCD1304Device, 64  
     TCD1304Device::Frame\_Header\_struct, 9  
 TRIGGER\_PIN  
     TCD1304Device2.h, 99  
 trigger\_pin  
     TCD1304Device, 64  
 trigger\_pin\_mode  
     TCD1304Device, 65  
 triggered\_read  
     TCD1304Device, 54, 55  
 triggerPin  
     TCD1304Device\_Controller\_251208.ino, 179  
 update\_read\_buffer  
     TCD1304Device, 56  
 usb\_string\_descriptor\_struct\_manufacturer, 65  
     bDescriptorType, 65  
     bLength, 65  
     wString, 65  
 usb\_string\_descriptor\_struct\_product, 66  
     bDescriptorType, 66

- bLength, [66](#)
  - wString, [66](#)
- usb\_string\_descriptor\_struct\_serial\_number, [66](#)
  - bDescriptorType, [66](#)
  - bLength, [66](#)
  - wString, [66](#)
- usb\_string\_manufacturer\_name
  - TCD1304Device\_Controller\_251208.ino, [179](#)
- usb\_string\_product\_name
  - TCD1304Device\_Controller\_251208.ino, [179](#)
- usb\_string\_serial\_number
  - TCD1304Device\_Controller\_251208.ino, [179](#)
- USBSPEED
  - TCD1304Device2.h, [100](#)
- USBTRANSFERSECS
  - TCD1304Device2.h, [100](#)
- versionstr
  - TCD1304Device\_Controller\_251208.ino, [179](#)
- VFS
  - TCD1304Device2.h, [100](#)
- VPERBIT
  - TCD1304Device2.h, [100](#)
- wait\_read
  - TCD1304Device, [56](#)
- wait\_triggered\_read
  - TCD1304Device, [57](#)
- wait\_triggers
  - TCD1304Device, [57](#)
- wordLength
  - parselib2.cpp, [77](#)
  - parselib2.h, [87](#)
- wString
  - usb\_string\_descriptor\_struct\_manufacturer, [65](#)
  - usb\_string\_descriptor\_struct\_product, [66](#)
  - usb\_string\_descriptor\_struct\_serial\_number, [66](#)