

hsk-libs-dev

257

Generated by Doxygen 1.8.12

Contents

1	HSK XC878 μC Library Developers' Manual	1
1.1	Preface	1
1.2	About This Document	1
1.2.1	Project Layout	1
2	The XC878 8-Bit Microcontroller Platform	5
2.1	Registers and Paging	5
2.2	Memory Limitations	6
2.2.1	Overlaying	6
2.3	Pointers	7
3	C51 Compiler Toolchain Setup	9
3.1	Device	9
3.2	Target	9
3.3	C51	10
3.4	LX51 Locate	11
3.5	LX51 Misc	12
3.6	Inline Assembler	13
3.7	Device Programming and Debugging	13
4	Using the Small Device C Compiler (SDCC)	15
4.1	Processor Architecture	15
4.2	SDCC Header File for XC878	15
4.3	Compiling Code	16
4.4	Linking	16
4.5	Programming	16
4.6	Memory Usage Compatibility	17
4.7	Interrupts	18

5	The Project Makefile	21
5.1	Generating the Documentation	21
5.1.1	Dependencies	21
5.2	Building	22
5.3	Implementation Details	22
5.3.1	Building	22
5.3.2	Documentation	22
5.3.3	Clean	23
5.4	Cygwin	23
6	Code Requirements	25
6.1	SFR Pages	25
6.2	ISRs	25
6.3	Memory	25
7	Variables and Memory	27
7.1	Implications of Overlaying	28
8	Coding Conventions and Guidelines	31
8.1	Code/Comment Indention and Formatting	31
8.2	General Comment Guidelines	32
8.2.1	List Formatting	32
8.2.2	Tables	33
8.2.3	Inline Comments	33
8.3	Function Documentation	34
8.3.1	Return Values	34
8.3.2	Public and Private Functions	34
8.4	Grouping Documentation	34
8.5	File Naming and Documentation	35
8.5.1	Headers	35
8.5.2	C Files	36
8.5.3	ISR Headers	37
8.6	Member Naming Conventions	37
8.6.1	Public Example	38
8.6.2	Defines	38

9 Authors	39
10 Deprecated List	41
11 Module Index	43
11.1 Modules	43
12 Data Structure Index	45
12.1 Data Structures	45
13 File Index	47
13.1 File List	47
14 Module Documentation	49
14.1 CAN Node Status Fields	49
14.1.1 Detailed Description	49
14.1.2 Macro Definition Documentation	49
14.1.2.1 CAN_STATUS_ALERT	49
14.1.2.2 CAN_STATUS_BOFF	50
14.1.2.3 CAN_STATUS_EWRN	50
14.1.2.4 CAN_STATUS_LEC	50
14.1.2.5 CAN_STATUS_RXOK	51
14.1.2.6 CAN_STATUS_TXOK	51
14.2 External Interrupt Channels	52
14.2.1 Detailed Description	52
14.2.2 Macro Definition Documentation	52
14.2.2.1 EX_EXINT0	52
14.2.2.2 EX_EXINT1	52
14.2.2.3 EX_EXINT2	53
14.2.2.4 EX_EXINT3	53
14.2.2.5 EX_EXINT4	53
14.2.2.6 EX_EXINT5	53
14.2.2.7 EX_EXINT6	53

14.3 External Interrupt Triggers	54
14.3.1 Detailed Description	54
14.3.2 Macro Definition Documentation	54
14.3.2.1 EX_EDGE_BOTH	54
14.3.2.2 EX_EDGE_DISABLE	54
14.3.2.3 EX_EDGE_FALLING	54
14.3.2.4 EX_EDGE_RISING	54
14.4 External Interrupt Input Ports	55
14.4.1 Detailed Description	56
14.4.2 Macro Definition Documentation	56
14.4.2.1 EX_EXINT0_P05	56
14.4.2.2 EX_EXINT0_P14	56
14.4.2.3 EX_EXINT1_P50	56
14.4.2.4 EX_EXINT1_P53	56
14.4.2.5 EX_EXINT2_P51	56
14.4.2.6 EX_EXINT2_P54	56
14.4.2.7 EX_EXINT3_P11	57
14.4.2.8 EX_EXINT3_P30	57
14.4.2.9 EX_EXINT3_P40	57
14.4.2.10 EX_EXINT3_P55	57
14.4.2.11 EX_EXINT4_P32	57
14.4.2.12 EX_EXINT4_P37	57
14.4.2.13 EX_EXINT4_P41	57
14.4.2.14 EX_EXINT4_P56	57
14.4.2.15 EX_EXINT5_P15	58
14.4.2.16 EX_EXINT5_P33	58
14.4.2.17 EX_EXINT5_P44	58
14.4.2.18 EX_EXINT5_P52	58
14.4.2.19 EX_EXINT6_P16	58
14.4.2.20 EX_EXINT6_P34	58

14.4.2.21 EX_EXINT6_P42	58
14.4.2.22 EX_EXINT6_P45	58
14.4.2.23 EX_EXINT6_P57	58
14.5 Input Port Access	59
14.5.1 Detailed Description	59
14.5.2 Macro Definition Documentation	59
14.5.2.1 IO_PORT_GET	59
14.5.2.2 IO_PORT_IN_INIT	60
14.5.2.3 IO_PORT_ON_GND	60
14.5.2.4 IO_PORT_ON_HIGH	60
14.6 Output Port Access	61
14.6.1 Detailed Description	61
14.6.2 Macro Definition Documentation	61
14.6.2.1 IO_PORT_DRAIN_DISABLE	61
14.6.2.2 IO_PORT_DRAIN_ENABLE	61
14.6.2.3 IO_PORT_OUT_INIT	62
14.6.2.4 IO_PORT_OUT_SET	62
14.6.2.5 IO_PORT_STRENGTH_STRONG	63
14.6.2.6 IO_PORT_STRENGTH_WEAK	63
14.7 I/O Port Pull-Up/-Down Setup	64
14.7.1 Detailed Description	64
14.7.2 Macro Definition Documentation	64
14.7.2.1 IO_PORT_PULL_DISABLE	64
14.7.2.2 IO_PORT_PULL_DOWN	64
14.7.2.3 IO_PORT_PULL_ENABLE	64
14.7.2.4 IO_PORT_PULL_INIT	65
14.7.2.5 IO_PORT_PULL_UP	65
14.8 Variable Access	66
14.8.1 Detailed Description	66
14.8.2 Macro Definition Documentation	66

14.8.2.1	IO_VAR_GET	66
14.8.2.2	IO_VAR_SET	66
14.9	Pulse Width Detection Units	68
14.9.1	Detailed Description	68
14.9.2	Macro Definition Documentation	68
14.9.2.1	PWC_UNIT_SUM_RAW	68
14.10	Pulse Width Times	69
14.10.1	Detailed Description	69
14.10.2	Macro Definition Documentation	69
14.10.2.1	PWC_UNIT_WIDTH_MS	69
14.10.2.2	PWC_UNIT_WIDTH_NS	69
14.10.2.3	PWC_UNIT_WIDTH_RAW	69
14.10.2.4	PWC_UNIT_WIDTH_US	69
14.11	Pulse Frequencies	70
14.11.1	Detailed Description	70
14.11.2	Macro Definition Documentation	70
14.11.2.1	PWC_UNIT_FREQ_H	70
14.11.2.2	PWC_UNIT_FREQ_M	70
14.11.2.3	PWC_UNIT_FREQ_S	70
14.12	Pulse Duty Times	71
14.12.1	Detailed Description	71
14.12.2	Macro Definition Documentation	71
14.12.2.1	PWC_UNIT_DUTYH_MS	71
14.12.2.2	PWC_UNIT_DUTYH_NS	71
14.12.2.3	PWC_UNIT_DUTYH_RAW	72
14.12.2.4	PWC_UNIT_DUTYH_US	72
14.12.2.5	PWC_UNIT_DUTYL_MS	72
14.12.2.6	PWC_UNIT_DUTYL_NS	72
14.12.2.7	PWC_UNIT_DUTYL_RAW	72
14.12.2.8	PWC_UNIT_DUTYL_US	72
14.13	SSC I/O Ports	73
14.13.1	Detailed Description	73
14.13.2	Macro Definition Documentation	73
14.13.2.1	SSC_MRST_P05	73
14.13.2.2	SSC_MRST_P14	74
14.13.2.3	SSC_MRST_P15	74
14.13.2.4	SSC_MTSR_P04	74
14.13.2.5	SSC_MTSR_P13	74
14.13.2.6	SSC_MTSR_P14	74
14.13.2.7	SSC_SCLK_P03	74
14.13.2.8	SSC_SCLK_P12	74
14.13.2.9	SSC_SCLK_P13	74

15 Data Structure Documentation	75
15.1 hsk_flash_struct Struct Reference	75
15.1.1 Detailed Description	76
15.1.2 Field Documentation	76
15.1.2.1 boot	76
15.1.2.2 error	76
15.1.2.3 hsk_flash_chksum	76
15.1.2.4 hsk_flash_prefix	76
15.1.2.5 reset	76
15.2 hsk_isr14_callback Struct Reference	77
15.2.1 Detailed Description	77
15.2.2 Field Documentation	78
15.2.2.1 NMIECC	78
15.2.2.2 NMIFLASH	78
15.2.2.3 NMIPLL	78
15.2.2.4 NMIVDDP	78
15.2.2.5 NMIWDT	78
15.3 hsk_isr5_callback Struct Reference	79
15.3.1 Detailed Description	80
15.3.2 Field Documentation	80
15.3.2.1 CANSRC0	80
15.3.2.2 CCTOVF	80
15.3.2.3 EOFSYN	80
15.3.2.4 ERRSYN	80
15.3.2.5 EXF2	81
15.3.2.6 NDOV	81
15.3.2.7 TF2	81
15.4 hsk_isr6_callback Struct Reference	81
15.4.1 Detailed Description	82
15.4.2 Field Documentation	82

15.4.2.1	ADCSR0	82
15.4.2.2	ADCSR1	82
15.4.2.3	CANSRC1	82
15.4.2.4	CANSRC2	83
15.5	hsk_isr8_callback Struct Reference	83
15.5.1	Detailed Description	84
15.5.2	Field Documentation	84
15.5.2.1	EOC	84
15.5.2.2	EXF2	84
15.5.2.3	EXINT2	85
15.5.2.4	IERR	85
15.5.2.5	IRDY	85
15.5.2.6	NDOV	85
15.5.2.7	RI	85
15.5.2.8	TF2	85
15.5.2.9	TI	85
15.6	hsk_isr9_callback Struct Reference	86
15.6.1	Detailed Description	86
15.6.2	Field Documentation	87
15.6.2.1	CANSRC3	87
15.6.2.2	EXINT3	87
15.6.2.3	EXINT4	87
15.6.2.4	EXINT5	87
15.6.2.5	EXINT6	87

16 File Documentation	89
16.1 config.h File Reference	89
16.1.1 Detailed Description	90
16.1.2 Macro Definition Documentation	90
16.1.2.1 CAN0_BAUD	90
16.1.2.2 CAN0_IO	90
16.1.2.3 CAN1_BAUD	90
16.1.2.4 CAN1_IO	90
16.1.2.5 CLK	90
16.2 hsk_adc/hsk_adc.c File Reference	91
16.2.1 Detailed Description	93
16.2.2 Macro Definition Documentation	93
16.2.2.1 ADC_CHANNELS	93
16.2.2.2 ADC_CLK_12MHz	93
16.2.2.3 ADC_CLK_6MHz	94
16.2.2.4 ADC_CLK_750kHz	94
16.2.2.5 ADC_CLK_8MHz	94
16.2.2.6 ADC_QUEUE	94
16.2.2.7 BIT_ADC_DIS	94
16.2.2.8 BIT_ANON	94
16.2.2.9 BIT_ASEN_PARALLEL	94
16.2.2.10 BIT_ASEN_SEQUENTIAL	94
16.2.2.11 BIT_CHNR	95
16.2.2.12 BIT CTC	95
16.2.2.13 BIT_DW	95
16.2.2.14 BIT_EMPTY	95
16.2.2.15 BIT_ENGT	95
16.2.2.16 BIT_FILL	95
16.2.2.17 BIT_IEN	95
16.2.2.18 BIT_IMODE	95

16.2.2.19	BIT_REQCHNR	96
16.2.2.20	BIT_RESULT	96
16.2.2.21	BIT_VFCTR	96
16.2.2.22	BIT_WFR	96
16.2.2.23	CNT_CHNR	96
16.2.2.24	CNT_CTC	96
16.2.2.25	CNT_FILL	96
16.2.2.26	CNT_REQCHNR	96
16.2.2.27	CNT_RESULT	97
16.2.3	Function Documentation	97
16.2.3.1	hsk_adc_close()	97
16.2.3.2	hsk_adc_disable()	97
16.2.3.3	hsk_adc_enable()	97
16.2.3.4	hsk_adc_init()	97
16.2.3.5	hsk_adc_isr10()	98
16.2.3.6	hsk_adc_isr8()	98
16.2.3.7	hsk_adc_isr_warmup10()	98
16.2.3.8	hsk_adc_open10()	99
16.2.3.9	hsk_adc_open8()	99
16.2.3.10	hsk_adc_request()	99
16.2.3.11	hsk_adc_service()	100
16.2.3.12	hsk_adc_warmup10()	100
16.2.4	Variable Documentation	101
16.2.4.1	nextChannel	101
16.2.4.2	ptr10	101
16.2.4.3	ptr8	101
16.2.4.4	targets	101
16.3	hsk_adc/hsk_adc.h File Reference	101
16.3.1	Detailed Description	103
16.3.2	Macro Definition Documentation	103

16.3.2.1	ADC_RESOLUTION_10	103
16.3.2.2	ADC_RESOLUTION_8	103
16.3.2.3	hsk_adc_open	103
16.3.2.4	hsk_adc_warmup	104
16.3.3	Typedef Documentation	104
16.3.3.1	hsk_adc_channel	104
16.3.4	Function Documentation	104
16.3.4.1	hsk_adc_close()	104
16.3.4.2	hsk_adc_disable()	104
16.3.4.3	hsk_adc_enable()	104
16.3.4.4	hsk_adc_init()	105
16.3.4.5	hsk_adc_open10()	105
16.3.4.6	hsk_adc_open8()	106
16.3.4.7	hsk_adc_request()	106
16.3.4.8	hsk_adc_service()	106
16.3.4.9	hsk_adc_warmup10()	107
16.4	hsk_boot/hsk_boot.c File Reference	107
16.4.1	Detailed Description	109
16.4.2	Macro Definition Documentation	110
16.4.2.1	BIT_EORDRES	110
16.4.2.2	BIT_EXTOSCR	110
16.4.2.3	BIT_MXB	110
16.4.2.4	BIT_MXB19	110
16.4.2.5	BIT_MXM	110
16.4.2.6	BIT_NDIVH	110
16.4.2.7	BIT_NDIVL	111
16.4.2.8	BIT_NMIPLL	111
16.4.2.9	BIT_OSCSS	111
16.4.2.10	BIT_PDIV	111
16.4.2.11	BIT_PLL_LOCK	111

16.4.2.12 BIT_PLLBYP	111
16.4.2.13 BIT_PLLPD	112
16.4.2.14 BIT_PLLR	112
16.4.2.15 BIT_PLLRDRES	112
16.4.2.16 BIT_XPD	112
16.4.2.17 CNT_MXB	112
16.4.2.18 CNT_NDIVH	112
16.4.2.19 CNT_NDIVL	112
16.4.2.20 CNT_PDIV	113
16.4.2.21 PDATA_PAGE	113
16.4.2.22 XRAM_BANK	113
16.4.2.23 XRAM_SELECTOR	113
16.4.3 Function Documentation	113
16.4.3.1 _sdcc_external_startup()	113
16.4.3.2 hsk_boot_extClock()	114
16.4.3.3 hsk_boot_io()	115
16.4.3.4 hsk_boot_isr_nmipll()	115
16.4.3.5 hsk_boot_mem()	115
16.4.4 Variable Documentation	115
16.4.4.1 boot	115
16.4.4.2 ndiv	115
16.4.4.3 pdiv	115
16.5 hsk_boot/hsk_boot.h File Reference	116
16.5.1 Detailed Description	116
16.5.2 Function Documentation	117
16.5.2.1 hsk_boot_extClock()	117
16.6 hsk_can/hsk_can.c File Reference	118
16.6.1 Detailed Description	124
16.6.2 The XC878 MultICAN Module	124
16.6.3 CAN List Management	125

16.6.4 Macro Definition Documentation	125
16.6.4.1 AUAD_DEC1	125
16.6.4.2 AUAD_INC1	126
16.6.4.3 AUAD_INC8	126
16.6.4.4 AUAD_OFF	126
16.6.4.5 BIT_ALIE	126
16.6.4.6 BIT_AM	126
16.6.4.7 BIT_AUAD	126
16.6.4.8 BIT_BRP	126
16.6.4.9 BIT_BSY	126
16.6.4.10 BIT_BUSY	127
16.6.4.11 BIT_CALM	127
16.6.4.12 BIT_CAN_DIS	127
16.6.4.13 BIT_CANDIS	127
16.6.4.14 BIT_CCE	127
16.6.4.15 BIT_DATA	127
16.6.4.16 BIT_DIR	127
16.6.4.17 BIT_DIV8	128
16.6.4.18 BIT_DLC	128
16.6.4.19 BIT_ERR	128
16.6.4.20 BIT_FCCFG	128
16.6.4.21 BIT_IDE	128
16.6.4.22 BIT_IDEXT	128
16.6.4.23 BIT_IDSTD	128
16.6.4.24 BIT_INIT	129
16.6.4.25 BIT_LECIE	129
16.6.4.26 BIT_LIST	129
16.6.4.27 BIT_MIDE	129
16.6.4.28 BIT_MMC	129
16.6.4.29 BIT_MSGVAL	129

16.6.4.30 BIT_NEWDAT	129
16.6.4.31 BIT_PRI	129
16.6.4.32 BIT_RBUSY	130
16.6.4.33 BIT_RWEN	130
16.6.4.34 BIT_RXEN	130
16.6.4.35 BIT_RXPND	130
16.6.4.36 BIT_RXSEL	130
16.6.4.37 BIT_RXUPD	130
16.6.4.38 BIT_SJW	130
16.6.4.39 BIT_TRIE	131
16.6.4.40 BIT_TSEG1	131
16.6.4.41 BIT_TSEG2	131
16.6.4.42 BIT_TXEN0	131
16.6.4.43 BIT_TXEN1	131
16.6.4.44 BIT_TXPND	131
16.6.4.45 BIT_TXRQ	131
16.6.4.46 CAN_AD_READ	132
16.6.4.47 CAN_AD_READY	132
16.6.4.48 CAN_AD_WRITE	132
16.6.4.49 CNT_AM	132
16.6.4.50 CNT_DATA	132
16.6.4.51 CNT_DLC	132
16.6.4.52 CNT_IDEXT	133
16.6.4.53 CNT_IDSTD	133
16.6.4.54 CNT_LIST	133
16.6.4.55 CNT_MMC	133
16.6.4.56 CNT_PRI	133
16.6.4.57 CNT_RXSEL	133
16.6.4.58 HSK_CAN_MSG_MAX	133
16.6.4.59 LIST_NODEx	133

16.6.4.60 LIST_PENDING	134
16.6.4.61 LIST_UNALLOC	134
16.6.4.62 MMC_DEFAULT	134
16.6.4.63 MMC_GATEWAYSRC	134
16.6.4.64 MMC_RXBASEFIFO	134
16.6.4.65 MMC_TXBASEFIFO	134
16.6.4.66 MMC_TXSLAVEFIFO	134
16.6.4.67 MOAMRn	134
16.6.4.68 MOARn	135
16.6.4.69 MOCTRn	135
16.6.4.70 MODATAHn	135
16.6.4.71 MODATALn	135
16.6.4.72 MOFCRn	135
16.6.4.73 MOFGPRn	135
16.6.4.74 MOFGPRn_BOT	135
16.6.4.75 MOFGPRn_CUR	135
16.6.4.76 MOFGPRn_SEL	136
16.6.4.77 MOFGPRn_TOP	136
16.6.4.78 MOSTATn	136
16.6.4.79 MOSTATn_PNEXT	136
16.6.4.80 NBTRx	136
16.6.4.81 NCRx	136
16.6.4.82 NECNTx	136
16.6.4.83 NFCRx	137
16.6.4.84 NIPRx	137
16.6.4.85 NPCRx	137
16.6.4.86 NSRx	137
16.6.4.87 OFF_LISTm	137
16.6.4.88 OFF_MOn	137
16.6.4.89 OFF_MSIDk	137

16.6.4.90 OFF_MSPNDk	137
16.6.4.91 OFF_NODEx	138
16.6.4.92 PAN_CMD_ALLOC	138
16.6.4.93 PAN_CMD_ALLOCBEFORE	138
16.6.4.94 PAN_CMD_ALLOCBEHIND	138
16.6.4.95 PAN_CMD_INIT	138
16.6.4.96 PAN_CMD_MOVE	138
16.6.4.97 PAN_CMD_MOVEBEFORE	138
16.6.4.98 PAN_CMD_MOVEBEHIND	138
16.6.4.99 PAN_CMD_NOP	139
16.6.4.100 PANAR1	139
16.6.4.101 PANAR2	139
16.6.4.102 PANCMD	139
16.6.4.103 PANCTR	139
16.6.4.104 PANCTR_READY	139
16.6.4.105 PANSTATUS	140
16.6.4.106 PRI_ID	140
16.6.4.107 PRI_LIST	140
16.6.4.108 RESET	140
16.6.4.109 RESET_DATA	140
16.6.4.110 SET	140
16.6.4.111 SET_DATA	140
16.6.5 Function Documentation	140
16.6.5.1 hsk_can_data_getIntelSignal()	140
16.6.5.2 hsk_can_data_getMotorolaSignal()	141
16.6.5.3 hsk_can_data_getSignal()	141
16.6.5.4 hsk_can_data_setIntelSignal()	142
16.6.5.5 hsk_can_data_setMotorolaSignal()	142
16.6.5.6 hsk_can_data_setSignal()	143
16.6.5.7 hsk_can_disable()	144

16.6.5.8	hsk_can_enable()	144
16.6.5.9	hsk_can_fifo_connect()	145
16.6.5.10	hsk_can_fifo_create()	145
16.6.5.11	hsk_can_fifo_delete()	146
16.6.5.12	hsk_can_fifo_disconnect()	146
16.6.5.13	hsk_can_fifo_getData()	147
16.6.5.14	hsk_can_fifo_getId()	148
16.6.5.15	hsk_can_fifo_move()	148
16.6.5.16	hsk_can_fifo_next()	148
16.6.5.17	hsk_can_fifo_setRxMask()	149
16.6.5.18	hsk_can_fifo_setupRx()	149
16.6.5.19	hsk_can_fifo_updated()	149
16.6.5.20	hsk_can_init()	150
16.6.5.21	hsk_can_msg_connect()	151
16.6.5.22	hsk_can_msg_create()	152
16.6.5.23	hsk_can_msg_delete()	152
16.6.5.24	hsk_can_msg_disconnect()	153
16.6.5.25	hsk_can_msg_getData()	154
16.6.5.26	hsk_can_msg_move()	154
16.6.5.27	hsk_can_msg_receive()	154
16.6.5.28	hsk_can_msg_send()	155
16.6.5.29	hsk_can_msg_sent()	155
16.6.5.30	hsk_can_msg_setData()	155
16.6.5.31	hsk_can_msg_updated()	155
16.6.5.32	hsk_can_status()	156
16.6.6	Variable Documentation	156
16.6.6.1	initialised	156
16.7	hsk_can/hsk_can.h File Reference	157
16.7.1	Detailed Description	159
16.7.2	CAN Message/Signal Tuples	159

16.7.3	CAN Node Management	160
16.7.4	Message Object Management	160
16.7.5	FIFOs	160
16.7.6	Message Data	161
16.7.7	Macro Definition Documentation	161
16.7.7.1	CAN0	161
16.7.7.2	CAN0_IO_P10_P11	161
16.7.7.3	CAN0_IO_P16_P17	161
16.7.7.4	CAN0_IO_P34_P35	161
16.7.7.5	CAN0_IO_P40_P41	161
16.7.7.6	CAN1	161
16.7.7.7	CAN1_IO_P01_P02	162
16.7.7.8	CAN1_IO_P14_P13	162
16.7.7.9	CAN1_IO_P32_P33	162
16.7.7.10	CAN_ENDIAN_INTEL	162
16.7.7.11	CAN_ENDIAN_MOTOROLA	162
16.7.7.12	CAN_ERROR	162
16.7.8	Typedef Documentation	163
16.7.8.1	hsk_can_fifo	163
16.7.8.2	hsk_can_msg	163
16.7.8.3	hsk_can_node	163
16.7.9	Function Documentation	163
16.7.9.1	hsk_can_data_getSignal()	163
16.7.9.2	hsk_can_data_setSignal()	164
16.7.9.3	hsk_can_disable()	165
16.7.9.4	hsk_can_enable()	165
16.7.9.5	hsk_can_fifo_connect()	165
16.7.9.6	hsk_can_fifo_create()	166
16.7.9.7	hsk_can_fifo_delete()	167
16.7.9.8	hsk_can_fifo_disconnect()	167

16.7.9.9	hsk_can_fifo_getData()	168
16.7.9.10	hsk_can_fifo_getId()	168
16.7.9.11	hsk_can_fifo_next()	169
16.7.9.12	hsk_can_fifo_setRxMask()	169
16.7.9.13	hsk_can_fifo_setupRx()	170
16.7.9.14	hsk_can_fifo_updated()	170
16.7.9.15	hsk_can_init()	171
16.7.9.16	hsk_can_msg_connect()	172
16.7.9.17	hsk_can_msg_create()	172
16.7.9.18	hsk_can_msg_delete()	173
16.7.9.19	hsk_can_msg_disconnect()	173
16.7.9.20	hsk_can_msg_getData()	174
16.7.9.21	hsk_can_msg_receive()	174
16.7.9.22	hsk_can_msg_send()	175
16.7.9.23	hsk_can_msg_sent()	175
16.7.9.24	hsk_can_msg_setData()	175
16.7.9.25	hsk_can_msg_updated()	175
16.7.9.26	hsk_can_status()	176
16.8	hsk_ex/hsk_ex.c File Reference	176
16.8.1	Detailed Description	178
16.8.2	Macro Definition Documentation	178
16.8.2.1	BIT_EXINT0	178
16.8.2.2	BIT_EXINT1	178
16.8.2.3	BIT_EXINT2	178
16.8.2.4	BIT_EXINT3	178
16.8.2.5	BIT_EXINT4	179
16.8.2.6	BIT_EXINT5	179
16.8.2.7	BIT_EXINT6	179
16.8.2.8	BIT_IMODE	179
16.8.2.9	CNT_EXINT	179

16.8.3	Function Documentation	179
16.8.3.1	hsk_ex_channel_disable()	179
16.8.3.2	hsk_ex_channel_enable()	180
16.8.3.3	hsk_ex_port_close()	180
16.8.3.4	hsk_ex_port_open()	180
16.8.4	Variable Documentation	180
16.8.4.1	modpiselBit	181
16.8.4.2	modpiselSel	181
16.8.4.3	portAltsel	181
16.8.4.4	portBit	181
16.8.4.5	ports	181
16.9	hsk_ex/hsk_ex.h File Reference	182
16.9.1	Detailed Description	184
16.9.2	Typedef Documentation	184
16.9.2.1	hsk_ex_channel	184
16.9.2.2	hsk_ex_port	184
16.9.3	Function Documentation	184
16.9.3.1	hsk_ex_channel_disable()	184
16.9.3.2	hsk_ex_channel_enable()	185
16.9.3.3	hsk_ex_port_close()	185
16.9.3.4	hsk_ex_port_open()	185
16.10	hsk_filter/hsk_filter.h File Reference	186
16.10.1	Detailed Description	186
16.10.2	Macro Definition Documentation	186
16.10.2.1	FILTER_FACTORY	186
16.10.2.2	FILTER_GROUP_FACTORY	187
16.11	hsk_flash/hsk_flash.c File Reference	187
16.11.1	Detailed Description	191
16.11.2	Flash Registers	191
16.11.3	Flash Timer	191

16.11.4 DPTR Byte Order	191
16.11.5 Macro Definition Documentation	191
16.11.5.1 ADDR_DFLASH	191
16.11.5.2 ADDR_PFLASH	191
16.11.5.3 ADDR_ROM	191
16.11.5.4 ADDR_XRAM	192
16.11.5.5 BIT_EEABORT	192
16.11.5.6 BIT_EEBSY	192
16.11.5.7 BIT_ERASE	192
16.11.5.8 BIT_FTEN	192
16.11.5.9 BIT_MAS1	192
16.11.5.10 BIT_MODE	192
16.11.5.11 BIT_NMIFLASH	193
16.11.5.12 BIT_NVSTR	193
16.11.5.13 BIT_OFVAL	193
16.11.5.14 BIT_PROG	193
16.11.5.15 BIT_YE	193
16.11.5.16 BYTES_PAGE_DFLASH	193
16.11.5.17 BYTES_PAGE_PFLASH	193
16.11.5.18 BYTES_WORDLINE_DFLASH	193
16.11.5.19 BYTES_WORDLINE_PFLASH	194
16.11.5.20 CNT_OFVAL	194
16.11.5.21 DPH	194
16.11.5.22 DPL	194
16.11.5.23 FREE_BEHIND	194
16.11.5.24 FREE_LATEST	194
16.11.5.25 FREE_NONE	195
16.11.5.26 LEN_DFLASH	195
16.11.5.27 LEN_PFLASH	195
16.11.5.28 LEN_ROM	195

16.11.5.29	LEN_XRAM	195
16.11.5.30	MOVCI	195
16.11.5.31	PAGE_FLASH	195
16.11.5.32	PAGE_RAM	195
16.11.5.33	STATE_DELETE	196
16.11.5.34	STATE_DETECT	196
16.11.5.35	STATE_IDLE	196
16.11.5.36	STATE_REQUEST	196
16.11.5.37	STATE_RESET	196
16.11.5.38	STATE_WRITE	196
16.11.5.39	VAR_ASM	196
16.11.5.40	VAR_AT	197
16.11.6	Function Documentation	197
16.11.6.1	hsk_flash_init()	197
16.11.6.2	hsk_flash_isr_nmiflash()	198
16.11.6.3	hsk_flash_write()	198
16.11.7	Variable Documentation	199
16.11.7.1	dflash	199
16.11.7.2	EECON	199
16.11.7.3	FCON	199
16.11.7.4	FCS	199
16.11.7.5	FCS1	199
16.11.7.6	FEAH	199
16.11.7.7	FEAL	200
16.11.7.8	FEALH	200
16.11.7.9	flash	200
16.11.7.10	flashDptr	200
16.11.7.11	free	200
16.11.7.12	FTVAL	200
16.11.7.13	dent	201

16.11.7.14atest	201
16.11.7.15oldest	201
16.11.7.16ptr	201
16.11.7.17size	201
16.11.7.18state	201
16.11.7.19wrap	201
16.11.7.20xdataDptr	202
16.12hsk_flash/hsk_flash.h File Reference	202
16.12.1 Detailed Description	203
16.12.2 Byte Order	204
16.12.3 Macro Definition Documentation	204
16.12.3.1 FLASH_PWR_FIRST	204
16.12.3.2 FLASH_PWR_ON	205
16.12.3.3 FLASH_PWR_RESET	205
16.12.3.4 FLASH_STRUCT_FACTORY	205
16.12.3.5 XC878_16FF	206
16.12.4 Function Documentation	206
16.12.4.1 hsk_flash_init()	206
16.12.4.2 hsk_flash_write()	207
16.13hsk_icm7228/hsk_icm7228.c File Reference	207
16.13.1 Detailed Description	208
16.13.2 Macro Definition Documentation	208
16.13.2.1 ILLUMINATE_OFFSET	208
16.13.3 Function Documentation	208
16.13.3.1 hsk_icm7228_illuminate()	208
16.13.3.2 hsk_icm7228_writeDec()	209
16.13.3.3 hsk_icm7228_writeHex()	209
16.13.3.4 hsk_icm7228_writeString()	210
16.13.4 Variable Documentation	210
16.13.4.1 codepage	210

16.14hsk_icm7228/hsk_icm7228.h File Reference	211
16.14.1 Detailed Description	212
16.14.2 Macro Definition Documentation	212
16.14.2.1 ICM7228_FACTORY	212
16.14.3 Function Documentation	213
16.14.3.1 hsk_icm7228_illuminate()	213
16.14.3.2 hsk_icm7228_writeDec()	213
16.14.3.3 hsk_icm7228_writeHex()	214
16.14.3.4 hsk_icm7228_writeString()	214
16.15hsk_io/hsk_io.h File Reference	215
16.15.1 Detailed Description	216
16.15.2 I/O Port Pull-Up/-Down Table	216
16.16hsk_isr/hsk_isr.c File Reference	217
16.16.1 Detailed Description	219
16.16.2 ISR Callback Reaction Time	219
16.16.3 Macro Definition Documentation	220
16.16.3.1 BIT_ADCSR0	220
16.16.3.2 BIT_ADCSR1	220
16.16.3.3 BIT_CANSRC0	220
16.16.3.4 BIT_CANSRC1	220
16.16.3.5 BIT_CANSRC2	220
16.16.3.6 BIT_CANSRC3	220
16.16.3.7 BIT_CCTOVF	221
16.16.3.8 BIT_EOC	221
16.16.3.9 BIT_EOFSYN	221
16.16.3.10BIT_ERRSYN	221
16.16.3.11BIT_EXF2 [1/2]	221
16.16.3.12BIT_EXF2 [2/2]	221
16.16.3.13BIT_EXINT2	221
16.16.3.14BIT_EXINT3	222

16.16.3.15	BIT_EXINT4	222
16.16.3.16	BIT_EXINT5	222
16.16.3.17	BIT_EXINT6	222
16.16.3.18	BIT_IERR	222
16.16.3.19	BIT_IRDY	222
16.16.3.20	BIT_NDOV [1/2]	222
16.16.3.21	BIT_NDOV [2/2]	223
16.16.3.22	BIT_NMIECC	223
16.16.3.23	BIT_NMIFLASH	223
16.16.3.24	BIT_NMIPLL	223
16.16.3.25	BIT_NMIVDDP	223
16.16.3.26	BIT_NMIWDT	223
16.16.3.27	BIT_RI	223
16.16.3.28	BIT_RMAP	224
16.16.3.29	BIT_TF2 [1/2]	224
16.16.3.30	BIT_TF2 [2/2]	224
16.16.3.31	BIT_TI	224
16.16.4	Function Documentation	224
16.16.4.1	dummy()	224
16.16.4.2	hsk_isr_root1()	224
16.16.4.3	ISR_hsk_isr14()	225
16.16.4.4	ISR_hsk_isr5()	225
16.16.4.5	ISR_hsk_isr6()	225
16.16.4.6	ISR_hsk_isr8()	226
16.16.4.7	ISR_hsk_isr9()	226
16.16.4.8	nmidummy()	226
16.16.5	Variable Documentation	227
16.16.5.1	hsk_isr14	227
16.16.5.2	hsk_isr5	227
16.16.5.3	hsk_isr6	227

16.16.5.4 hsk_isr8	227
16.16.5.5 hsk_isr9	227
16.17 hsk_isr/hsk_isr.h File Reference	228
16.17.1 Detailed Description	229
16.17.2 SFR Pages	229
16.17.3 Register Banks	229
16.17.4 Variable Documentation	230
16.17.4.1 hsk_isr14	230
16.17.4.2 hsk_isr5	230
16.17.4.3 hsk_isr6	230
16.17.4.4 hsk_isr8	230
16.17.4.5 hsk_isr9	230
16.18 hsk_pwc/hsk_pwc.c File Reference	231
16.18.1 Detailed Description	234
16.18.2 Macro Definition Documentation	234
16.18.2.1 BIT_CCM0	234
16.18.2.2 BIT_CCTBx	234
16.18.2.3 BIT_CCTOVEN	234
16.18.2.4 BIT_CCTOVF	234
16.18.2.5 BIT_CCTPRE	235
16.18.2.6 BIT_CCTST	235
16.18.2.7 BIT_IMODE	235
16.18.2.8 BIT_T2CCFG	235
16.18.2.9 BIT_T2CCU_DIS	235
16.18.2.10 BIT_TIMSYN	235
16.18.2.11 CHAN_BUF_SIZE	235
16.18.2.12 CNT_CCMx	236
16.18.2.13 CNT_EXINTx	236
16.18.2.14 EDGE_DEFAULT_MODE	236
16.18.2.15 PWC_CC0_EXINT_BIT	236

16.18.2.16	PWC_CC0_EXINT_REG	236
16.18.2.17	PWC_CC1_EXINT_BIT	236
16.18.2.18	PWC_CC1_EXINT_REG	236
16.18.2.19	PWC_CC2_EXINT_BIT	236
16.18.2.20	PWC_CC2_EXINT_REG	237
16.18.2.21	PWC_CC3_EXINT_BIT	237
16.18.2.22	PWC_CC3_EXINT_REG	237
16.18.2.23	PWC_CHANNELS	237
16.18.3	Function Documentation	237
16.18.3.1	hsk_pwc_ccn()	237
16.18.3.2	hsk_pwc_channel_captureMode()	237
16.18.3.3	hsk_pwc_channel_close()	238
16.18.3.4	hsk_pwc_channel_edgeMode()	238
16.18.3.5	hsk_pwc_channel_getValue()	238
16.18.3.6	hsk_pwc_channel_open()	239
16.18.3.7	hsk_pwc_channel_trigger()	239
16.18.3.8	hsk_pwc_disable()	240
16.18.3.9	hsk_pwc_enable()	240
16.18.3.10	hsk_pwc_init()	240
16.18.3.11	hsk_pwc_isr_cc0_p30()	241
16.18.3.12	hsk_pwc_isr_cc0_p40()	241
16.18.3.13	hsk_pwc_isr_cc0_p55()	242
16.18.3.14	hsk_pwc_isr_cc1_p32()	242
16.18.3.15	hsk_pwc_isr_cc1_p41()	242
16.18.3.16	hsk_pwc_isr_cc1_p56()	243
16.18.3.17	hsk_pwc_isr_cc2_p33()	243
16.18.3.18	hsk_pwc_isr_cc2_p44()	243
16.18.3.19	hsk_pwc_isr_cc2_p52()	244
16.18.3.20	hsk_pwc_isr_cc3_p34()	244
16.18.3.21	hsk_pwc_isr_cc3_p45()	244

16.18.3.22	hsk_pwc_isr_cc3_p57()	245
16.18.3.23	hsk_pwc_isr_ccn()	245
16.18.3.24	hsk_pwc_isr_cctOverflow()	245
16.18.3.25	hsk_pwc_port_open()	245
16.18.4	Variable Documentation	246
16.18.4.1	averageOver	246
16.18.4.2	buffer	246
16.18.4.3	channels	247
16.18.4.4	hsk_pwc_ports	247
16.18.4.5	inBit	247
16.18.4.6	inCount	247
16.18.4.7	inSel	247
16.18.4.8	invalid	247
16.18.4.9	lastCapture	248
16.18.4.10	overflow	248
16.18.4.11	overflows	248
16.18.4.12	portBit	248
16.18.4.13	portSel	248
16.18.4.14	pos	248
16.18.4.15	prescaler	248
16.18.4.16	state	249
16.18.4.17	sum	249
16.19	hsk_pwc/hsk_pwc.h File Reference	249
16.19.1	Detailed Description	252
16.19.2	Macro Definition Documentation	252
16.19.2.1	PWC_CC0	252
16.19.2.2	PWC_CC0_P30	252
16.19.2.3	PWC_CC0_P40	252
16.19.2.4	PWC_CC0_P55	253
16.19.2.5	PWC_CC1	253

16.19.2.6 PWC_CC1_P32	253
16.19.2.7 PWC_CC1_P41	253
16.19.2.8 PWC_CC1_P56	253
16.19.2.9 PWC_CC2	253
16.19.2.10 PWC_CC2_P33	253
16.19.2.11 PWC_CC2_P44	253
16.19.2.12 PWC_CC2_P52	254
16.19.2.13 PWC_CC3	254
16.19.2.14 PWC_CC3_P34	254
16.19.2.15 PWC_CC3_P45	254
16.19.2.16 PWC_CC3_P57	254
16.19.2.17 PWC_EDGE_BOTH	254
16.19.2.18 PWC_EDGE_FALLING	254
16.19.2.19 PWC_EDGE_RISING	254
16.19.2.20 PWC_MODE_EXT	255
16.19.2.21 PWC_MODE_SOFT	255
16.19.3 Typedef Documentation	255
16.19.3.1 hsk_pwc_channel	255
16.19.3.2 hsk_pwc_port	255
16.19.4 Function Documentation	255
16.19.4.1 hsk_pwc_channel_captureMode()	255
16.19.4.2 hsk_pwc_channel_close()	256
16.19.4.3 hsk_pwc_channel_edgeMode()	256
16.19.4.4 hsk_pwc_channel_getValue()	256
16.19.4.5 hsk_pwc_channel_open()	257
16.19.4.6 hsk_pwc_channel_trigger()	257
16.19.4.7 hsk_pwc_disable()	258
16.19.4.8 hsk_pwc_enable()	258
16.19.4.9 hsk_pwc_init()	258
16.19.4.10 hsk_pwc_port_open()	259

16.20hsk_pwm/hsk_pwm.c File Reference	260
16.20.1 Detailed Description	262
16.20.2 Macro Definition Documentation	262
16.20.2.1 BIT_CCU_DIS	262
16.20.2.2 BIT_CCUCCFG	263
16.20.2.3 BIT_ECT13O	263
16.20.2.4 BIT_PSL	263
16.20.2.5 BIT_PSL63	263
16.20.2.6 BIT_TnCLK	263
16.20.2.7 BIT_TnMODEN	263
16.20.2.8 BIT_TnRR	263
16.20.2.9 BIT_TnRS	263
16.20.2.10BIT_TnSTR	264
16.20.2.11CNT_MSEL6n	264
16.20.2.12CNT_PSL	264
16.20.2.13CNT_TnCLK	264
16.20.2.14CNT_TnMODEN	264
16.20.2.15MOD_MSEL6n	264
16.20.3 Function Documentation	264
16.20.3.1 hsk_pwm_channel_set()	264
16.20.3.2 hsk_pwm_disable()	265
16.20.3.3 hsk_pwm_enable()	265
16.20.3.4 hsk_pwm_init()	265
16.20.3.5 hsk_pwm_outChannel_dir()	266
16.20.3.6 hsk_pwm_port_close()	266
16.20.3.7 hsk_pwm_port_open()	267
16.20.4 Variable Documentation	267
16.20.4.1 ports	267
16.20.4.2 pos	268
16.20.4.3 sel	268

16.21 hsk_pwm/hsk_pwm.h File Reference	268
16.21.1 Detailed Description	270
16.21.2 Macro Definition Documentation	271
16.21.2.1 PWM_60	271
16.21.2.2 PWM_61	271
16.21.2.3 PWM_62	271
16.21.2.4 PWM_63	271
16.21.2.5 PWM_CC60	271
16.21.2.6 PWM_CC61	272
16.21.2.7 PWM_CC62	272
16.21.2.8 PWM_COUT60	272
16.21.2.9 PWM_COUT61	272
16.21.2.10 PWM_COUT62	272
16.21.2.11 PWM_COUT63	272
16.21.2.12 PWM_OUT_60_P30	272
16.21.2.13 PWM_OUT_60_P31	272
16.21.2.14 PWM_OUT_60_P40	273
16.21.2.15 PWM_OUT_60_P41	273
16.21.2.16 PWM_OUT_61_P00	273
16.21.2.17 PWM_OUT_61_P01	273
16.21.2.18 PWM_OUT_61_P31	273
16.21.2.19 PWM_OUT_61_P32	273
16.21.2.20 PWM_OUT_61_P33	273
16.21.2.21 PWM_OUT_61_P44	273
16.21.2.22 PWM_OUT_61_P45	274
16.21.2.23 PWM_OUT_62_P04	274
16.21.2.24 PWM_OUT_62_P05	274
16.21.2.25 PWM_OUT_62_P34	274
16.21.2.26 PWM_OUT_62_P35	274
16.21.2.27 PWM_OUT_62_P46	274

16.21.2.28 PWM_OUT_62_P47	274
16.21.2.29 PWM_OUT_63_P03	274
16.21.2.30 PWM_OUT_63_P37	275
16.21.2.31 PWM_OUT_63_P43	275
16.21.3 Typedef Documentation	275
16.21.3.1 hsk_pwm_channel	275
16.21.3.2 hsk_pwm_outChannel	275
16.21.3.3 hsk_pwm_port	275
16.21.4 Function Documentation	275
16.21.4.1 hsk_pwm_channel_set()	275
16.21.4.2 hsk_pwm_disable()	276
16.21.4.3 hsk_pwm_enable()	276
16.21.4.4 hsk_pwm_init()	276
16.21.4.5 hsk_pwm_outChannel_dir()	277
16.21.4.6 hsk_pwm_port_close()	277
16.21.4.7 hsk_pwm_port_open()	278
16.22 hsk_ssc/hsk_ssc.c File Reference	278
16.22.1 Detailed Description	280
16.22.2 Macro Definition Documentation	280
16.22.2.1 BIT_CIS	280
16.22.2.2 BIT_EIR	280
16.22.2.3 BIT_EIREN	280
16.22.2.4 BIT_EN	281
16.22.2.5 BIT_LB	281
16.22.2.6 BIT_MIS	281
16.22.2.7 BIT_MS	281
16.22.2.8 BIT_RIR	281
16.22.2.9 BIT_RIREN	281
16.22.2.10 BIT_RMAP	281
16.22.2.11 BIT_SIS	282

16.22.2.12BIT_SSC_DIS	282
16.22.2.13BIT_TIR	282
16.22.2.14BIT_TIREN	282
16.22.2.15CNT_SEL	282
16.22.3 Function Documentation	282
16.22.3.1 hsk_ssc_disable()	282
16.22.3.2 hsk_ssc_enable()	282
16.22.3.3 hsk_ssc_init()	282
16.22.3.4 hsk_ssc_ports()	283
16.22.3.5 hsk_ssc_talk()	283
16.22.3.6 ISR_hsk_ssc()	284
16.22.4 Variable Documentation	284
16.22.4.1 bufState	284
16.22.4.2 rcount	284
16.22.4.3 rptr	284
16.22.4.4 wcount	284
16.22.4.5 wptr	284
16.23hsk_ssc/hsk_ssc.h File Reference	285
16.23.1 Detailed Description	286
16.23.2 Half Duplex Operation	287
16.23.3 Macro Definition Documentation	287
16.23.3.1 hsk_ssc_busy	287
16.23.3.2 SSC_BAUD	287
16.23.3.3 SSC_CONF	288
16.23.3.4 SSC_MASTER	288
16.23.3.5 SSC_SLAVE	288
16.23.4 Function Documentation	288
16.23.4.1 hsk_ssc_disable()	288
16.23.4.2 hsk_ssc_enable()	289
16.23.4.3 hsk_ssc_init()	289

16.23.4.4 hsk_ssc_ports()	289
16.23.4.5 hsk_ssc_talk()	289
16.24hsk_timers/hsk_timer01.c File Reference	290
16.24.1 Detailed Description	291
16.24.2 Macro Definition Documentation	292
16.24.2.1 BIT_ET0	292
16.24.2.2 BIT_ET1	292
16.24.2.3 BIT_RMAP	292
16.24.2.4 BIT_T0M	292
16.24.2.5 BIT_T1M	292
16.24.2.6 CNT_T0M	292
16.24.2.7 CNT_T1M	292
16.24.3 Function Documentation	292
16.24.3.1 hsk_timer01_setup()	292
16.24.3.2 hsk_timer0_disable()	293
16.24.3.3 hsk_timer0_enable()	293
16.24.3.4 hsk_timer0_setup()	293
16.24.3.5 hsk_timer1_disable()	294
16.24.3.6 hsk_timer1_enable()	294
16.24.3.7 hsk_timer1_setup()	294
16.24.3.8 ISR_hsk_timer0()	295
16.24.3.9 ISR_hsk_timer1()	295
16.24.4 Variable Documentation	295
16.24.4.1 callback	295
16.24.4.2 overflow	295
16.24.4.3 timers	295
16.25hsk_timers/hsk_timer01.h File Reference	296
16.25.1 Detailed Description	297
16.25.2 Function Documentation	297
16.25.2.1 hsk_timer0_disable()	297

16.25.2.2 hsk_timer0_enable()	297
16.25.2.3 hsk_timer0_setup()	297
16.25.2.4 hsk_timer1_disable()	298
16.25.2.5 hsk_timer1_enable()	298
16.25.2.6 hsk_timer1_setup()	298
16.26hsk_wdt/hsk_wdt.c File Reference	299
16.26.1 Detailed Description	300
16.26.2 Watchdog Timer Registers	300
16.26.3 Macro Definition Documentation	300
16.26.3.1 BIT_WDTEN	300
16.26.3.2 BIT_WDTIN	300
16.26.3.3 BIT_WDTRS	300
16.26.4 Function Documentation	301
16.26.4.1 hsk_wdt_disable()	301
16.26.4.2 hsk_wdt_enable()	301
16.26.4.3 hsk_wdt_init()	301
16.26.4.4 hsk_wdt_service()	301
16.27hsk_wdt/hsk_wdt.h File Reference	302
16.27.1 Detailed Description	302
16.27.2 Hazards	303
16.27.3 Function Documentation	303
16.27.3.1 hsk_wdt_disable()	303
16.27.3.2 hsk_wdt_enable()	303
16.27.3.3 hsk_wdt_init()	303
16.27.3.4 hsk_wdt_service()	304
16.28main.c File Reference	304
16.28.1 Detailed Description	305
16.28.2 Macro Definition Documentation	306
16.28.2.1 PERSIST_VERSION	306
16.28.3 Function Documentation	306

16.28.3.1 init()	306
16.28.3.2 main()	307
16.28.3.3 p1_illuminate()	308
16.28.3.4 p1_init()	310
16.28.3.5 p1_refresh()	310
16.28.3.6 p1_writeDec()	310
16.28.3.7 p1_writeHex()	311
16.28.3.8 p1_writeString()	311
16.28.3.9 run()	312
16.28.3.10 tick0()	313
16.28.4 Variable Documentation	313
16.28.4.1 adc7	313
16.28.4.2 p1_buffer	313
16.28.4.3 persist	313
16.28.4.4 tick0_count_20	313
16.28.4.5 tick0_count_250	313
Index	315

Chapter 1

HSK XC878 μ C Library Developers' Manual

1.1 Preface

Welcome to the High Speed Karlsruhe (HSK) XC878 microcontroller (μ C) developers' manual. This document is intended for those who want to perform library development.

This document contains all the library header and code documentation.

See also

[PDF Version](#)

1.2 About This Document

This document is work in progress, so far the documentation for the libraries is mostly complete. Documentation of implemented applications is less so and like the applications still subject to a lot of change.

1.2.1 Project Layout

- `LICENSE.md`
 - ISCL and 3rd party licensing
- `Makefile`
 - Makefile to invoke the SDCC and doxygen toolchain
- `Makefile.local`
 - Local non-revisioned Makefile for overriding default parameters
- `README.md`
 - Repository README
- `uVisionupdate.sh`
 - Updates the μ Vision project's overlaying instructions

- `bin.c51/`
 - C51 toolchain output produced by Keil μ Vision (safe to delete)
- `bin.sdcc/`
 - SDCC compiler output (safe to delete)
- `conf/`
 - Project configuration files
- `conf/doxygen.common`
 - Basic doxygen settings
- `conf/doxygen.dbc`
 - Doxygen setting changes to create documentation from DBC headers
- `conf/doxygen.dev`
 - Doxygen setting changes to create the developer documentation
- `conf/doxygen.scripts`
 - Doxygen setting changes to create the scripts documentation
- `conf/doxygen.user`
 - Doxygen setting changes to create the user documentation
- `conf/sdcc`
 - SDCC configuration, contains basic CFLAGS and invokes version specific platform hacks
- `doc/`
 - Documentation build directory
- `gen/`
 - Generated code e.g. the `.mk` files with build instructions
- `gen/dbc/`
 - C headers generated from Vector DBCs (via `scripts/dbc2c.awk`)
- `gh-pages/`
 - Project documentation, published at <https://lonkamikaze.github.io/hsk-libs>
- `gh-pages/contrib/`
 - This directory contains 3rd party documentation
- `gh-pages/contrib/ICM7228.pdf`
 - Intersil ICM7228 8-Digit, LED Display Decoder Driver data sheet
- `gh-pages/contrib/Microcontroller-XC87x-Data-Sheet-V15-infineon.pdf`
 - Data sheet for the Infineon XC87x series
- `gh-pages/contrib/XC878_um_v1_1.pdf`
 - Infineon XC878 User Manual Version 1.1
- `hacks/`
 - Storage directory for hacks that are pulled in depending on platform parameters like the SDCC version

- `img/`
 - Pictures included in this documentation
- `inc/`
 - 3rd party headers
- `scripts/`
 - Contains build scripts used by the `Makefile`, this folder is documented in the a dedicated document
- `src/`
 - The project source code
- `src/doc/`
 - This directory contains general documentation that is not specific to a library, application or a file, i.e. this chapter of the documentation
- `src/hsk_.../`
 - Directories with this prefix contain library code
- `uVision/`
 - ARM Keil μ Vision project files

Chapter 2

The XC878 8-Bit Microcontroller Platform

The XC878 is an Intel 8051/8052 compatible μ C architecture. This entails strong memory limitations with severe implications to writing code.

The strength of the architecture is that the controller contains many specialized modules that, once set up, perform many tasks without intense interaction.

Critical for this project are the following kinds of modules:

- 10-Bit AD conversion channels
- Timers that can be triggered by external signals or perform PWM
- CAN controller

See also

XC878 Reference Manual: [XC878_um_v1_1.pdf](#)

ARM Keil Infineon XC878-16FF page: <http://www.keil.com/dd/chip/4480.htm>

Infineon XC87x Series Overview: <http://www.infineon.com/cms/en/product/microcontrollers/8-bit/html?channel=db3a304323b87bc20123dcee653f7007&tab=2>

8051 Basics Tutorial: <http://www.8052.com/tut8051>

2.1 Registers and Paging

The XC878 functions and modules are controlled through so called Special Function Registers (SFRs).

Due to the number of modules and functions of the controller a lot more registers are present than the 128 that can be addressed. These 128 registers are addressed in the upper directly addressable address range from $0x80$ to $0xFF$.

To circumvent the 128 register limit each functional block of registers has a paging register that can be used to access different Pages of registers. In C code this is done using the `SFR_PAGE()` macro defined in the Infineon/XC878.h header file that is provided by Keil μ Vision, the IDE used for this project or the headers can directly be [downloaded from ARM](#).

Paging only affects code that directly interacts with the hardware. One of the benefits of *using* these libraries is that paging is not an issue in the logical code.

Each section of the [XC878 Reference Manual](#) has a Register Overview that contains a table of pages and registers.

2.2 Memory Limitations

The 8051 platform offers 128 bytes of `data` memory in the address range 0x00-0x7F in front of the SFR address range. Because 128 bytes are insufficient, the 8051 architecture knows several kinds of memory that are accessed in different manners and thus quicker or slower to access.

The 8052 has an additional 128 bytes of indirectly addressable memory. This memory is accessed through the key word `idata`. Access to `idata` is slower than to `data`. The syntax for declaring a variable in `idata` memory is:

```
<type> idata <identifier> [= <value>;
```

The additional `idata` memory is located in the upper half of the address range. The lower half accesses `data` memory. Any `data` access to a pointer actually is `idata` access. This is why SFRs cannot be accessed with pointers. They are masked by `idata` memory.

The slowest kind of memory used by this library is the `xdata` memory. The `xdata` memory makes 3kb of additional memory available and the libraries place all large data structures in them.

Variables are declared in `xdata` with the following syntax:

```
<type> xdata <identifier> [= <value>;
```

The first 256 bytes of `xdata` memory are also accessible as 8 bit addressed `pdata`. Using `pdata` is faster than `xdata`. The `p` in `pdata` stands for paged. Historically the 8052 family of μ Cs used register P2 for paging. The XC878 instead provides an SFR named XADDRH.

However current 8051 C compilers don't support paging. I.e. one would have to ensure that structs and buffers do not cross page borders and update XADDRH manually. So instead of making the code more complicated and messing with the linkers XADDRH is fixed to the first `xdata` page and `pdata` is simply used as an additional 256 bytes of relatively fast memory.

There also is a 128 bits wide memory range of `bit` variables, which is used by single bit variables of the type `bool`.

In contrast to the small amount of available RAM, 64k of ROM are available to hold executable code. Thus a program well designed to the XC878 is one that produces a lot of static code to reduce the required amount of runtime memory use.

Code resides in its own address range, the `code` block. The μ C can be run from code residing in `xdata` as well, to bootstrap the μ C. Constants can also be placed in the `code` block.

2.2.1 Overlaying

In order to mitigate the memory limitations of the platform the C51 and SDCC compilers perform an optimisation called overlaying.

The compilers build a call graph, much like the one in the documentation of [main\(\)](#). The call graph is a directed graph and functions (i.e. their local variables and parameters) may occupy the same space in memory, provided they cannot reach each other in the graph.

E.g. [main\(\)](#) can reach both [init\(\)](#) and [run\(\)](#), thus [main\(\)](#) may not occupy the same memory as [init\(\)](#) and [run\(\)](#). However [run\(\)](#) and [init\(\)](#) cannot reach each other, so they may store their data in the same memory.

This reduces the use of the stack, which is expensive in terms of runtime (this statement is not generally true, it just applies to the 8051 family of μ Cs).

Functions may not be called more than once at a time. I.e. they may not be recursive or called from regular code and interrupts both. Both compilers provide a `reentrant` keyword to make functions operate on the stack. Its use should be avoided if possible.

Both C51 and SDCC do not track function pointers. Thus creating a function pointer results in a false call in the call tree. A call through a function pointer is not added to the call tree.

The C51 tool chain offers [call tree manipulations](#) and SDCC provides the [nooverlay pragma](#) to mitigate this.

The section about [Implications of Overlaying](#) lists best practices to optimize code for overlaying.

2.3 Pointers

Due to the existence of different kinds of memory, pointers come in two variations, generic pointers and memory-specific pointers. Generic pointers take 3 bytes of memory and are by far the slowest to process. Memory-specific pointers take 1 byte for `data`, `idata` or `pdata` and 2 bytes for `xdata` or `code`. They are also faster to process.

Note that the `data` keyword needs to be explicitly specified for `data` pointers. Pointers declared without explicit mention of the memory type always result in generic pointers.

Pointers can be stored in different kinds of memory than the memory they point to:

```
<type> <ptr_target_mem> * <ptr_mem> <identifier>;
```

The following example creates an `idata` pointer to a struct in `xdata` memory:

```
struct foo xdata * idata p_foo;
```


Chapter 3

C51 Compiler Toolchain Setup

This section describes the necessary compiler toolchain setup based on the Keil μ Vision IDE.

3.1 Device

It is critical for device flashing and programming to select the correct version of the μ C. This dialogue also allows you to select the extended linker and assembler. Doing so is imperative to perform the necessary link time optimisations to fit the libraries into the limited memory of the device.

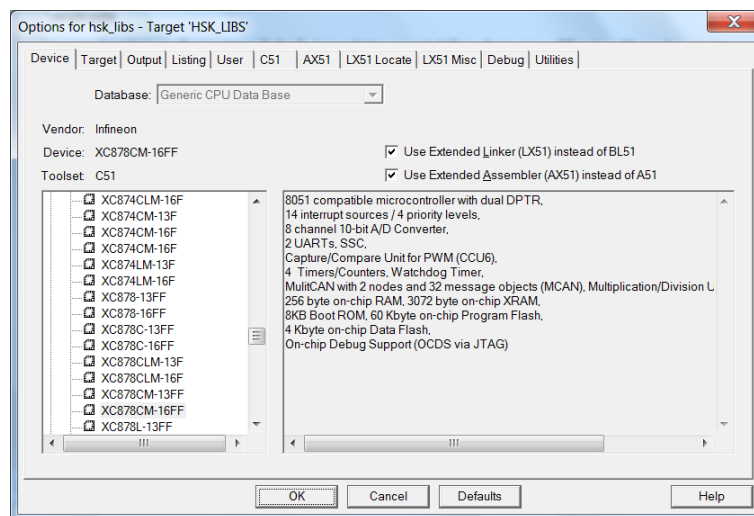


Figure 3.1 Keil μ Vision Device options dialogue

3.2 Target

The target dialogue lets you select several CPU architecture and memory layout settings.

The following options need to be set:

- Xtal (MHz):

- This needs to be set to your external oscillator frequency, otherwise flashing and debugging might be unreliable
- Memory Model: Small
 - This setting means that variables are by default assigned to the first 128 bytes of directly addressable RAM, variables can still be mapped to different memory sections manually as described in [Memory Limitations](#)
- Code ROM Size: Large: 64K program
 - This setting allows up to 64k of program data to be written to the device
- Use On-chip ROM
- Use On-chip XRAM
- Use multiple DPTR registers
 - This allows the compiler to reduce address writes of reoccurring pointer targets by using multiple pointer registers
- Safe address extension SFR in interrupts
 - XRAM/xdata access is not atomic. Thus interrupts using XRAM can interrupt and corrupt XRAM access of functions. This setting preserves the XRAM address registers and thus protects them from corruption

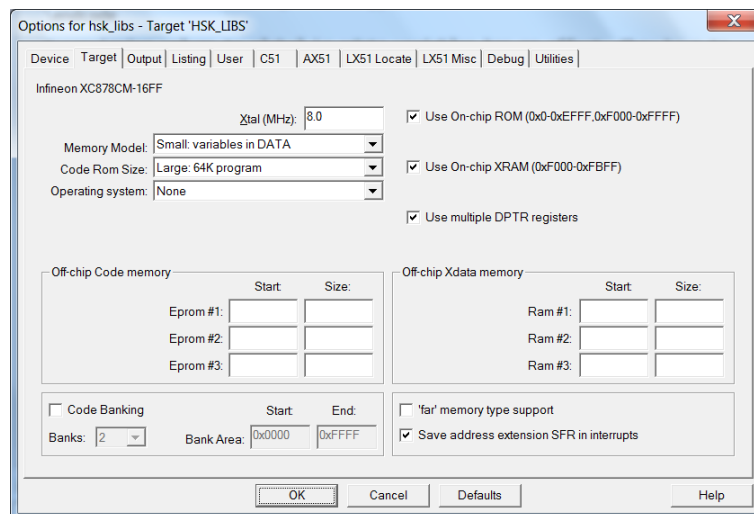


Figure 3.2 Keil µVision Target options dialogue

3.3 C51

The C51 is the C compiler configuration dialogue. The following settings are not obligatory for use of the HSK libraries, but recommended.

- Preprocessor Symbols
 - Define: `__xdata`, `__pdata`, `__idata`
 - * This input field allows passing on preprocessor definitions to the preprocessor
 - * The empty `__xdata`, `__pdata`, `__idata` defines allow C51 to ignore SDCC style memory assignments, this is useful to make such assignments where C51 does not support them

- Code Optimization
 - Level: 11: Reuse Common Exit Code
 - * The highest level of optimisation, allowing the compiler the largest reduction of memory use
 - * Select 4 or lower for debugging, all the common code eliminations prevent the debugger from mapping large chunks of C code to assembler code, making the program flow difficult to follow
 - Emphasis: Favor speed
 - * Surprisingly this often produces smaller code than the `favor size` setting
 - Global Register Coloring
 - * This setting allows the compiler to optimise register use throughout the entire application, reducing memory use and improving performance
 - Linker Code Packing
 - * Activates a link time optimisation, after linking the application, the linker will replace long distance jumps with short jumps where applicable
- Warnings: Warninglevel 2
- Enable ANSI integer promotion rules

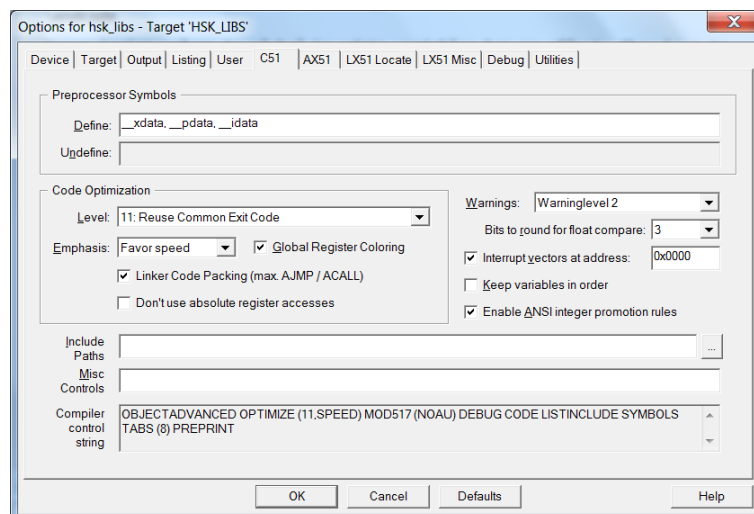


Figure 3.3 Keil µVision C51 options dialogue

3.4 LX51 Locate

LX51 is the extended linker of the C51 compiler tool chain, the Locate dialogue is used to map memory ranges. The form can also be used to assign portions of code to fixed addresses.

- User Memory Layout from Target Dialog
 - This option assigns the XC878 memory types to the appropriate address ranges
- User Classes: PDATA (X:0xF000-X:0xF0FF)
 - This option maps the `pdata` memory into the first 256 bytes of `xdata`

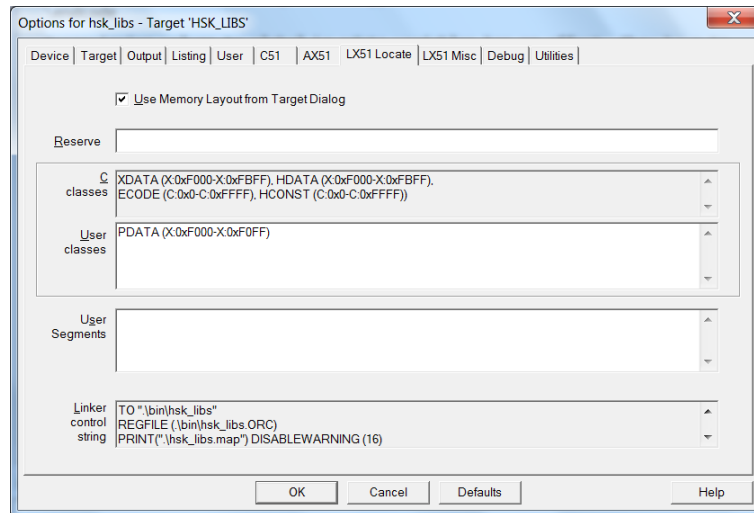


Figure 3.4 Keil µVision LX51 Locate options dialogue

3.5 LX51 Misc

The Misc dialogue holds the remaining linker settings.

- Overlay
 - This field can be used to add calls through function pointers to the call tree as is necessary for callback functions, the syntax is described in the µVision Help section OVERLAY Linker Directive
 - Manually filling this field can be avoided by running the `uVisionupdate.sh` script
- Misc controls: REMOVEUNUSED
 - This linker flag saves memory by discarding unused functions

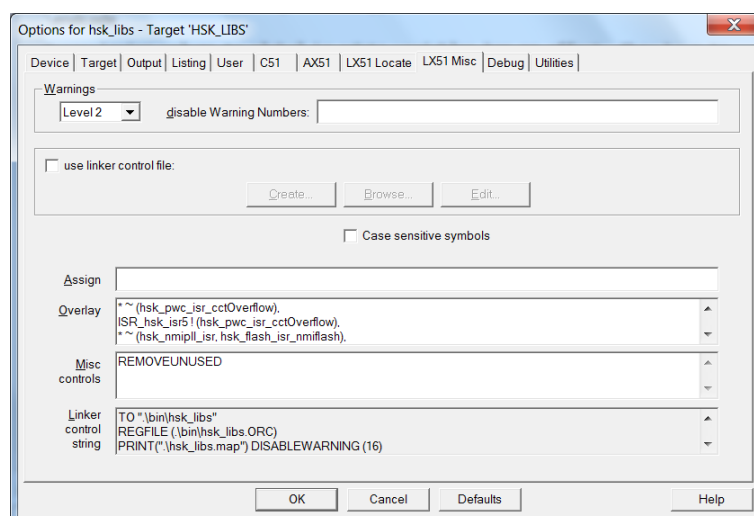


Figure 3.5 Keil µVision LX51 Misc options dialogue

3.6 Inline Assembler

Inline assembler has to be activated for Groups or single files individually. The Group Options can be found in the context menu of a Group.

To activate an option in this menu it needs to be unchecked and checked again.

- Generate Assembler SRC File
 - This option causes the compiler to generate assembler code instead of an object file
- Assemble SRC File
 - This option causes the compiler to assemble the generated assembler code

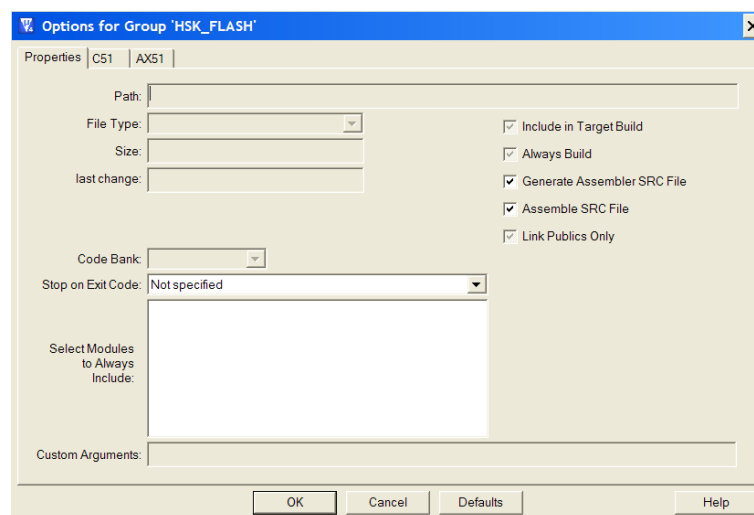


Figure 3.6 Keil μVision Group options dialogue

3.7 Device Programming and Debugging

To program or debug the device via On Chip Debug Support (OCDS) the Infineon Direct Access Server (DAS), a hardware access middleware, is required. The programming and debugging options can also be selected from the target options. Use the following settings in the Utilities tab:

- Use Target Driver for Flash Programming
 - Infineon DAS Client for XC800

Chapter 4

Using the Small Device C Compiler (SDCC)

This section describes how to compile code for the XC878 with the Small Device C Compiler. SDCC is an open source compiler supporting several 8 bit architectures.

This section is about using the compiler and maintaining C51 compatibility. Refer to [The Project Makefile](#) to build this project using SDCC.

See also

SDCC project: <http://sdcc.sf.net>
Small Device C Compiler Manual: [sdccman.pdf](#)

4.1 Processor Architecture

The 8051 architecture is selected with the parameter `-mmcs51`. Additionally the compiler needs to be invoked with the correct memory architecture, XRAM starts at address 0xF000 and is 3kb wide. For this the parameters `--xram-loc` and `--xram-size` are provided:

```
-mmcs51 --xram-loc 0xF000 --xram-size 3072
```

4.2 SDCC Header File for XC878

This projects includes the file `Infineon/XC878.h` in the `inc/` directory, which contains the SFR definitions for the XC878. It is a modified version of the `XC878.h` file provided by Keil μ Vision, which in turn is a Dave generated file.

The modification is an `#ifdef SDCC` block, with some compatibility glue to allow using the C51 code mostly unmodified.

To make the header available to `sdcc` the `inc/` should be added to the include search path with the `-I` parameter:

```
-mmcs51 --xram-loc 0xF000 --xram-size 3072 -I inc/
```

4.3 Compiling Code

Code is compiled using the `-c` parameter:

```
sdcc -mmcs51 --xram-loc 0xF000 --xram-size 3072 -I inc/ -o builddir/ -c example.c
```

The compiler will generate a number of files in `builddir`, among them `example.asm` and `example.rel`, the object file.

Instead of the build dir the output parameter `-o` can also take the name of the object file as a parameter.

SDCC can only compile one `.c` file at a time, thus every `.c` file must be compiled separately and linked in a separate step later.

4.4 Linking

Linking can be done by giving all required object files as parameters. The output file name will be based on the first input file or can explicitly be stated:

```
sdcc -mmcs51 --xram-loc 0xF000 --xram-size 3072 -I inc/ -o builddir/ -c example.rel lib1.rel lib2.rel
```

The output file would be `builddir/example.ihx` (Intel HEX). `-o builddir/example.hex` can be used to change the filename suffix to `.hex`, which is more convenient when using XC800_FLOAD to flash the μ C.

4.5 Programming

Whereas μ Vision as an IDE covers flashing, SDCC users need a separate tool to do so. One such tool is Infineon's XC800_FLOAD. FLOAD also requires DAS.

The use of FLOAD is very straightforward, make the following settings:

- Protocol: JTAG/SPD
- Physical Interface: UDAS/JTAG over USB
- Target Device: XC87x-16FF

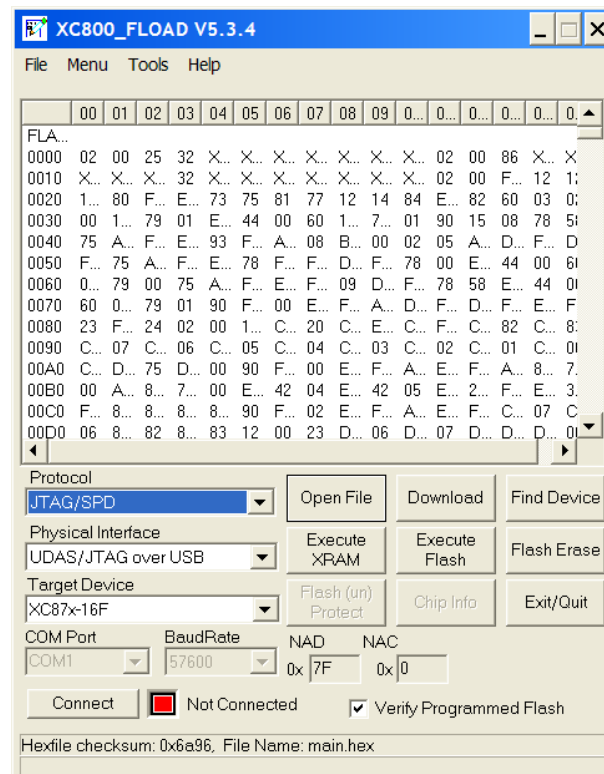


Figure 4.1 XC800_FLOAD Flash Programming Tool

The important functions of FLOAD are:

- Open: Open a HEX file for programming
- Download: Download the program to the μ C flash
- Flash Erase: Clear the μ C flash memory

See also

FLOAD download: <http://www.infineon.com/cms/en/product/microcontrollers/development-tools/html?channel=db3a304319c6f18c011a0b54923431e5>
 Infineon DAS Tool Interface website: <http://www.infineon.com/das/>

4.6 Memory Usage Compatibility

In order to write compatible code, the compatibility glue in the Infineon/XC878.h header maps keywords like `data`, `idata`, `xdata` etc. to their SDCC equivalents `__data`, `__idata` and `__xdata`.

Starting with SDCC 3.x C51 `code` and SDCC `__code` are largely interchangeable. This wasn't always the case, which results in two different styles for using `code`.

E.g. putting a variable into `code` space, C51 style:

```
ubyte code foo = 0x2A;
```

SDCC style:

```
const ubyte foo = 0x2A;
```

The C51 style is consistent with other variable space assignments and portable (it did not work with SDCC 2.x, but works with 3.x). The SDCC style is more logical in terms of writing code that communicates what one intends to do. The recommendation within this project is to use a redundant style, which also worked with SDCC 2.x with some C macro magic:

```
const ubyte code foo = 0x2A;
```

Function pointers are a special case. To SDCC all function pointers refer to `code`, C51 uses (slower, larger) generic pointers if the `code` keyword is missing. Unfortunately there is no compatible syntax to place `code` in a function pointer declaration. This can be circumvented with preprocessor instructions:

```
/*
 * SDCC does not like the \c code keyword for function pointers, C51 needs it
 * or it will use generic pointers.
 */
#ifdef SDCC
    #undef code
    #define code
#endif /* SDCC */
```

A function pointer declaration may look like this:

```
void (code *foo)(void);
```

Take care to restore `code` before the end of a `.h` file:

```
/*
 * Restore the usual meaning of \c code.
 */
#ifdef SDCC
    #undef code
    #define code    __code
#endif /* SDCC */
```

4.7 Interrupts

The most significant difference between interrupt handling in SDCC and C51 is that prototypes for the interrupts must be visible in the context of the `main()` function.

These prototypes can be enclosed in an `#ifdef`:

```
#ifndef _HSK_ISR_ISR_
#define _HSK_ISR_ISR_
void ISR_hsk_isr5(void) interrupt 5 using 1;
void ISR_hsk_isr6(void) interrupt 6 using 1;
void ISR_hsk_isr8(void) interrupt 8 using 1;
void ISR_hsk_isr9(void) interrupt 9 using 1;
void ISR_hsk_isr14(void) interrupt 14 using 2;
#endif /* _HSK_ISR_ISR_ */
```

In this project the issue is solved by placing the prototypes in a dedicated file with the file name suffix `.isr`. It is then included in the relevant headers in the following fashion:


```
/*
 * Required for SDCC to propagate ISR prototypes.
 */
#ifdef SDCC
#include "../hsk_isr/hsk_isr.isr"
#endif /* SDCC */
```

By not placing the ISR prototypes directly in the header file, unwanted and otherwise unnecessary inclusion of headers can be avoided. Only the small file with the ISR prototypes needs to be included in header files of ISR using code.

According to the SDCC manual functions called by ISRs must be reentrant or protected from memory overlay. This is done with a compiler instruction:

```
#pragma save
#pragma nooverlay
void isr_callback(void) using 1 {
    [...]
}
#pragma restore
```

Unfortunately this causes a compiler warning when using C51:

```
..\src\main.c(49): warning C245: unknown #pragma, line ignored
```

This can be avoided by making nooverlay conditional:

```
#pragma save
#ifdef SDCC
#pragma nooverlay
#endif
void isr_callback(void) using 1 {
    [...]
}
#pragma restore
```


Chapter 5

The Project Makefile

The project Makefile offers access to all the UNIX command line facilities of the project. The file is written for the FreeBSD make, which is a descendant of PMake. Some convenience and elegance was sacrificed to make the Makefile GNU Make compatible.

5.1 Generating the Documentation

The Makefile can invoke Doxygen with the `make` targets `html` and `pdf`:

```
# make html pdf
Searching for include files...
Searching for example files...
Searching for images...
[...]
```

The `html` target creates the directories `html/user/` and `html/dev/`, which contain the HTML version of this documentation.

The `pdf` target creates the directory `pdf/` with the PDF versions of this documentation.

The targets create a Users' and a Developers' Manual. The first only includes documentation for public interfaces (i.e. headers). The second also includes the documentation of the implementation and some additional tidbits in this chapter that are only of interest when developing the libraries instead of building applications with them.

5.1.1 Dependencies

In order to build the documentation the following tools need to be installed on the system:

- Doxygen
- GraphViz (for creating dependency graphs)
- TeX (for pdf_latex)

5.2 Building

The Makefile uses SDCC to build. This can be changed in the first lines of the Makefile. The default target `build` builds all the `.c` files. Each `.c` file containing a `main()` function will also be linked, resulting in a `.hex` file:

```
# make
sdcc -mmcs51 [...] -o bin.sdcc/hsk_adc/hsk_adc.rel -c src/hsk_adc/hsk_adc.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_boot/hsk_boot.rel -c src/hsk_boot/hsk_boot.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_can/hsk_can.rel -c src/hsk_can/hsk_can.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_wdt/hsk_wdt.rel -c src/hsk_wdt/hsk_wdt.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_icm7228/hsk_icm7228.rel -c src/hsk_icm7228/hsk_icm7228.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_isr/hsk_isr.rel -c src/hsk_isr/hsk_isr.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_pwc/hsk_pwc.rel -c src/hsk_pwc/hsk_pwc.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_pwm/hsk_pwm.rel -c src/hsk_pwm/hsk_pwm.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_timers/hsk_timer01.rel -c src/hsk_timers/hsk_timer01.c
sdcc -mmcs51 [...] -o bin.sdcc/hsk_flash/hsk_flash.rel -c src/hsk_flash/hsk_flash.c
sdcc -mmcs51 [...] -o bin.sdcc/main.rel -c src/main.c
sdcc -mmcs51 [...] -o bin.sdcc/main.hex bin.sdcc/hsk_timers/hsk_timer01.rel [...]
```

All compiler output is dumped into the `bin.sdcc/` directory. All the `.c` files are built, independent of whether they are linked into a `.hex` file.

5.3 Implementation Details

The Makefile consists of three parts:

- Building
- Documentation building
- Cleaning

5.3.1 Building

The Makefile declares the target `build` first to make it the default target. None of the build targets are manually defined. Instead with every invocation of `make` the script `scripts/build.sh` is invoked to regenerate the file `build.mk`.

The `build.sh` script searches the source directory for `.c` files and runs `scripts/depends.awk` in `-compile` mode to generate a dependency tree.

In the next stage `build.sh` generates the build instructions for each `.c` file.

The last step is to create the linking instructions. For that the script searches for `.c` files that appear to contain a `main()` function. The `scripts/depends.awk` script in `-link` mode is used to determine all the libraries that have to be linked with each of the `main()` containing `.c` files.

5.3.2 Documentation

The targets `doc` and `doc-private` build the user and the developer documentation. The `html` target simply copies them to the `html/` directory.

The `pdf` target copies the PDF versions of the manuals to `pdf/`, but first a PDF needs to be generated by running `make` in the `doc/latex/` and `doc-private/latex/` directories, which is done by the respective targets.

5.3.3 Clean

The `clean-doc` target removes the directory `doc/`, the `clean-doc-private` target removes the directory `doc-private/` and the target `clean-build` removes the directory `BUILDDIR`, which defaults to `bin.sdcc/`.

The meta-target `clean` invokes all these targets.

5.4 Cygwin

Using a combination of native Binaries and Cygwin, the complete set of build and generator facilities can be used from Microsoft Windows.

The following downloads are required:

- GIT: <https://git-scm.com/downloads>
- SDCC: <https://sourceforge.net/projects/sdcc/files/>
- Cygwin installer: <http://cygwin.com/install.html>

Additionally to the defaults the following Cygwin packages have to be installed:

- Devel: gcc-core
- Devel: libiconv
- Devel: make

After the installation the `PATH` variable should reference SDCC and Cygwin:

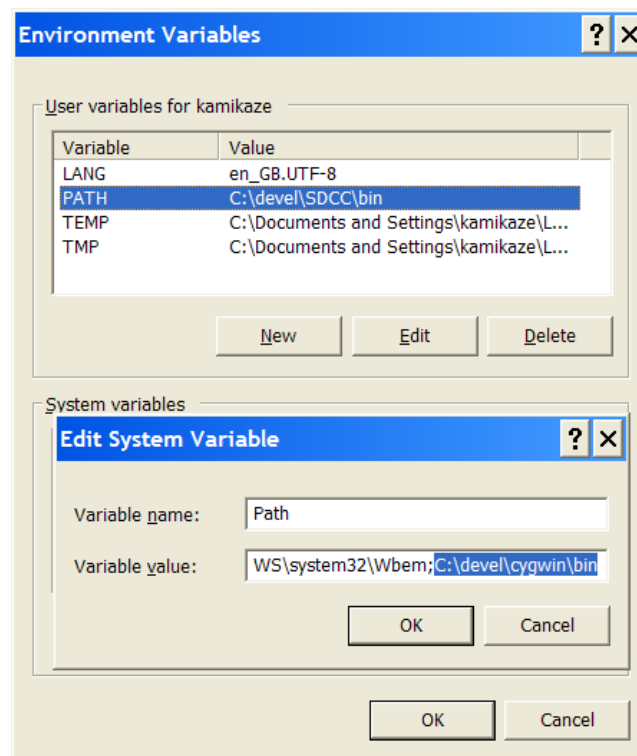
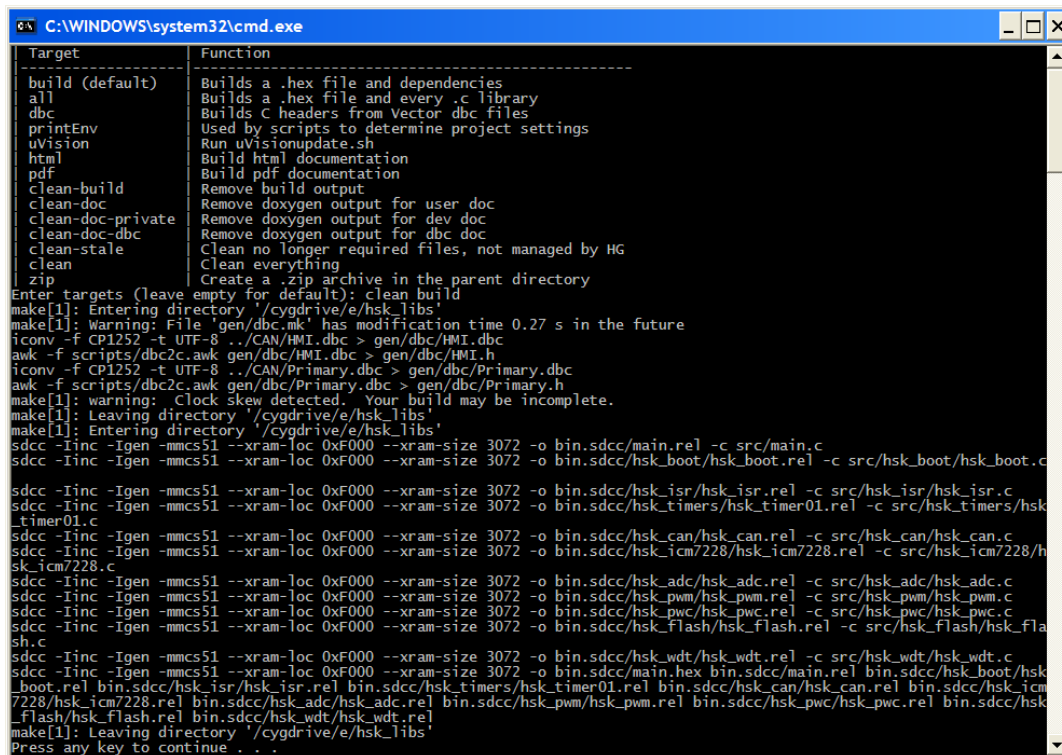


Figure 5.1 Cygwin and SDCC PATH environment

The libraries provide simple batch files to execute `uVisionupdate.sh` and call `make`. For ease of use the table of make targets is displayed:



```

C:\WINDOWS\system32\cmd.exe
Target      Function
-----
build (default)  Builds a .hex file and dependencies
all            Builds a .hex file and every .c library
dbc           Builds C headers from Vector dbc files
printEnv      Used by scripts to determine project settings
uVision       Run uVisionupdate.sh
html          Build html documentation
pdf           Build pdf documentation
clean-build   Remove build output
clean-doc     Remove doxygen output for user doc
clean-doc-private Remove doxygen output for dev doc
clean-doc-dbc Remove doxygen output for dbc doc
clean-stale   Clean no longer required files, not managed by HG
clean         Clean everything
zip           Create a .zip archive in the parent directory

Enter targets (leave empty for default): clean build
make[1]: Entering directory '/cygdrive/e/hsk_libs'
make[1]: Warning: File 'gen/dbc.mk' has modification time 0.27 s in the future
iconv -f CP1252 -t UTF-8 ../CAN/HMI.dbc > gen/dbc/HMI.dbc
awk -f scripts/dbc2c.awk gen/dbc/HMI.dbc > gen/dbc/HMI.h
iconv -f CP1252 -t UTF-8 ../CAN/Primary.dbc > gen/dbc/Primary.dbc
awk -f scripts/dbc2c.awk gen/dbc/Primary.dbc > gen/dbc/Primary.h
make[1]: warning: Clock skew detected. Your build may be incomplete.
make[1]: Leaving directory '/cygdrive/e/hsk_libs'
make[1]: Entering directory '/cygdrive/e/hsk_libs'
sdcc -Iinc -Igen -mmcs51 --xram-loc 0xF000 --xram-size 3072 -o bin.sdcc/main.rel -c src/main.c
sdcc -Iinc -Igen -mmcs51 --xram-loc 0xF000 --xram-size 3072 -o bin.sdcc/hsk_boot/hsk_boot.rel -c src/hsk_boot/hsk_boot.c
sdcc -Iinc -Igen -mmcs51 --xram-loc 0xF000 --xram-size 3072 -o bin.sdcc/hsk_isr/hsk_isr.rel -c src/hsk_isr/hsk_isr.c
sdcc -Iinc -Igen -mmcs51 --xram-loc 0xF000 --xram-size 3072 -o bin.sdcc/hsk_timers/hsk_timer01.rel -c src/hsk_timers/hsk_timer01.c
sdcc -Iinc -Igen -mmcs51 --xram-loc 0xF000 --xram-size 3072 -o bin.sdcc/hsk_can/hsk_can.rel -c src/hsk_can/hsk_can.c
sdcc -Iinc -Igen -mmcs51 --xram-loc 0xF000 --xram-size 3072 -o bin.sdcc/hsk_icm7228/hsk_icm7228.rel -c src/hsk_icm7228/hsk_icm7228.c
sdcc -Iinc -Igen -mmcs51 --xram-loc 0xF000 --xram-size 3072 -o bin.sdcc/hsk_adc/hsk_adc.rel -c src/hsk_adc/hsk_adc.c
sdcc -Iinc -Igen -mmcs51 --xram-loc 0xF000 --xram-size 3072 -o bin.sdcc/hsk_pwm/hsk_pwm.rel -c src/hsk_pwm/hsk_pwm.c
sdcc -Iinc -Igen -mmcs51 --xram-loc 0xF000 --xram-size 3072 -o bin.sdcc/hsk_pwc/hsk_pwc.rel -c src/hsk_pwc/hsk_pwc.c
sdcc -Iinc -Igen -mmcs51 --xram-loc 0xF000 --xram-size 3072 -o bin.sdcc/hsk_flash/hsk_flash.rel -c src/hsk_flash/hsk_flash.c
sdcc -Iinc -Igen -mmcs51 --xram-loc 0xF000 --xram-size 3072 -o bin.sdcc/hsk_wdt/hsk_wdt.rel -c src/hsk_wdt/hsk_wdt.c
sdcc -Iinc -Igen -mmcs51 --xram-loc 0xF000 --xram-size 3072 -o bin.sdcc/main.hex bin.sdcc/main.rel bin.sdcc/hsk_boot/hsk_boot.rel bin.sdcc/hsk_isr/hsk_isr.rel bin.sdcc/hsk_timers/hsk_timer01.rel bin.sdcc/hsk_can/hsk_can.rel bin.sdcc/hsk_icm7228/hsk_icm7228.rel bin.sdcc/hsk_adc/hsk_adc.rel bin.sdcc/hsk_pwm/hsk_pwm.rel bin.sdcc/hsk_pwc/hsk_pwc.rel bin.sdcc/hsk_flash/hsk_flash.rel bin.sdcc/hsk_wdt/hsk_wdt.rel
make[1]: Leaving directory '/cygdrive/e/hsk_libs'
Press any key to continue . . .

```

Figure 5.2 Make and available targets in Cygwin

Chapter 6

Code Requirements

To use these libraries the utilizing code must meet a small number of requirements.

6.1 SFR Pages

All public functions expect all pages set to 0 and reset all pages they touch to 0 before they exit.

All public functions also expect RMAP 0.

6.2 ISRs

The [hsk_isr.h](#) documentation lists the rules that need to be obeyed when implementing ISRs and callback functions:

- [SFR Pages](#)
- [Register Banks](#)

6.3 Memory

In order to access `pdata` and `xdata` the `hsk_boot` library must be linked.

Chapter 7

Variables and Memory

The Infineon/XC878.h header defines some unsigned data types:

- bool (1 bit)
- ulong (32 bits)
- uword (16 bits)
- ubyte (8 bits)

Signed types should only be used when necessary, floating point arithmetic should be avoided if in any way possible.

The correct place to store a variable depends on four factors:

- Size
- Lifetime
- Frequency of use during lifetime
- Overlay possibility

Size, and frequency are the most obvious, considering [Memory Limitations](#). It is desirable to use fast memory for frequently accessed variables. Large data structures like buffers, arrays and structs simply use too much of the precious memory space to put them anywhere but `xdata`.

Both the C51 and SDCC compilers use a technique called overlay to fit all variables into memory. A stack is only used in reentrant functions. Only `data/idata` variables can be stacked with `push/pop` instructions. Stacking `xdata` is emulated in software and thus very slow.

The overlay approach is to build a call graph and thus decide which variables are never used at the same time. These variables are mapped to the same fixed memory addresses.

Variables with a long lifetime are locals in the [main\(\)](#) function, static variables and global variables. These variables have to keep their state during the entire runtime. Thus they use memory space that cannot be shared with other variables.

ISRs and functions called by ISRs also cannot share memory, because there is no sensible way to make sure that a given function is not running when an interrupt occurs. In technical terms, each ISR is the root node of its own call graph.

The following table lists recommended memory types:

Context	Size	Critical	Memory
*	bool	*	bit
parameter	*	*	data
const	*	*	code
local	byte, word	*	data, idata
	>= long	no	xdata
		ISR/blocking	pdata, xdata
static/global	*	no	xdata
		ISR/blocking	pdata, xdata

ISR/blocking refers to memory accessed by ISRs, functions called back by ISRs and sections of code that block an interrupt.

7.1 Implications of Overlaying

First and foremost, [Overlaying](#) is only performed for the default memory. This project is built around the small memory model, i.e. only `data` memory is overlaid by SDCC and C51.

The `data` memory is only 128 bytes minus the used register banks large. With three register banks that means only 104 bytes of `data` memory are available. Thus non-overlayable variables should be placed in `idata` in order to use `data` memory for well overlayable variables.

Furthermore SDCC does not build a complete call tree, so it cannot eliminate unused functions like C51/LX51 and only overlays leaf functions. I.e. only functions that call no other functions are overlaid.

For SDCC overlaying ISRs is not possible, that is why locals of ISRs should in most cases be placed in `idata`, despite the performance impact.

A limitation of C51/LX51 is that it ignores explicit memory assignments in function parameters and always places them in the default memory, if they cannot be passed in registers. This limitation does not appear to be documented, but it was reported by an ARM support employee in case #530915.

SDCC does not share this limitation, it can place function parameters in all kinds of memory. Because such assignments are ignored by C51/LX51, parameter memory type should be optimised for SDCC. Explicit memory assignments prevent parameters from being passed in registers. SDCC only passes the first non-`bool` parameter in registers, so optimizations should be performed on the non-overlayable arguments following it.

For single use functions like `init` and `enable` functions, it might make sense to pass parameters in `xdata`. In such cases the SDCC memory assignment style should be used, to give the C51 preprocessor the chance to remove the assignments.

```
void hsk_can_init(const ubyte pins, const ulong __xdata baud);
```

If an application runs out of `data` space locals should be put into `idata` memory. Variables accessed by ISRs/ISR callbacks should be placed in `pdata` or `idata`. If that suffices the code should be checked for frequently accessed variables. The most frequent ones should be assigned to the default memory type if possible.

Both SDCC and C51 provide detailed information about memory use. The effects of relocating variables are often counter-intuitive, because it may interfere with several compiler and linker optimizations. This it is necessary to make use of this information in order to make sure that changes have the desired effect.

For SDCC check the assembler output for the DSEG area (search regex `/\\\.area DSEG/`):

```

;-----
; internal ram data
;-----
        .area DSEG      (DATA)
_hsk_adc_init_resolution_1_71:
        .ds 1
_hsk_adc_init_etc_1_72:
        .ds 1
_hsk_adc_init_sloc0_1_0:
        .ds 2
_hsk_adc_init_sloc1_1_0:
        .ds 2

```

The `.map` file lists the complete memory layout produced by the linker (search regex `/^DSEG/`):

Area	Addr	Size	Decimal Bytes (Attributes)
DSEG	00000000	00000080 =	128. bytes (REL,CON)

Value	Global	Global Defined In Module
00000018	_hsk_pwm_init_PARM_2	hsk_pwm
0000001C	_hsk_pwm_channel_set_PARM_2	hsk_pwm
0000001E	_hsk_pwm_channel_set_PARM_3	hsk_pwm
00000025	_hsk_icm7228_writeDec_PARM_2	hsk_icm7228
00000027	_hsk_icm7228_writeDec_PARM_3	hsk_icm7228
00000028	_hsk_icm7228_writeDec_PARM_4	hsk_icm7228
...		

In μ Vision the `.map` file can be accessed by double clicking the project in the project tree view (search string "D A T A"):

START	STOP	LENGTH	ALIGN	RELOC	MEMORY CLASS	SEGMENT NAME
=====						
* * * * * D A T A M E M O R Y * * * * *						
000000H	000007H	000008H	---	AT..	DATA	"REG BANK 0"
000008H	00000FH	000008H	---	AT..	DATA	"REG BANK 1"
000010H	000017H	000008H	---	AT..	DATA	"REG BANK 2"
000018H	00001BH	000004H	BYTE	UNIT	IDATA	_IDATA_GROUP_
00001CH.0	00001FH.7	000004H.0	---	---	**GAP**	
000020H.0	000020H.4	000000H.5	BIT	UNIT	BIT	_BIT_GROUP_
000020H.5	000020H.5	000000H.1	BIT	UNIT	BIT	?BI?HSK_CAN
000020H.6	000020H	000000H.2	---	---	**GAP**	
000021H	00003FH	00001FH	BYTE	UNIT	DATA	_DATA_GROUP_
000040H	000040H	000001H	BYTE	UNIT	IDATA	?STACK

See also

Overlaying in the SDCC manual

Global Registers used for Parameter Passing in the SDCC manual

Chapter 8

Coding Conventions and Guidelines

This section describes the coding style used in these libraries.

The term *member* in this section applies to functions, globals, structs, unions, typedefs and defines of the current library.

8.1 Code/Comment Indention and Formatting

Use 8 spaces wide real tabs for indention. Don't put spaces before a tab, even in comments, unless it is in a code or verbatim section.

In the following example tabs are symbolised by `<tab>` and the beginning of a line by `^`:

```
^/**
^ * Foobar struct.
^ */
^struct {
^<tab>/**
^<tab> * Lame text.
^<tab> *
^<tab> * Lame example:
^<tab> * \code
^<tab> * void foo() {
^<tab> * <tab>doSomething();
^<tab> * }
^<tab> * \endcode
^<tab> *
^<tab> * @see
^<tab> *<tab>Something else
^<tab> */
^<tab>ubyte member;
^} foobar;
```

Formatting on the other hand should be done using spaces. This way, no matter the displayed tab width, formatted code and comments will always look as intended.

The values assigned to preprocessor defines should be aligned. The recommended indention is 4 spaces behind the longest identifier in the file. All other defines should be aligned to the same column.

In the next example `FOOBAR` is the longest identifier and thus dictates the formatting of all values:

```
#define FOO      1
#define BAR      2
#define FOOBAR   3
#define ZOOM     4
```

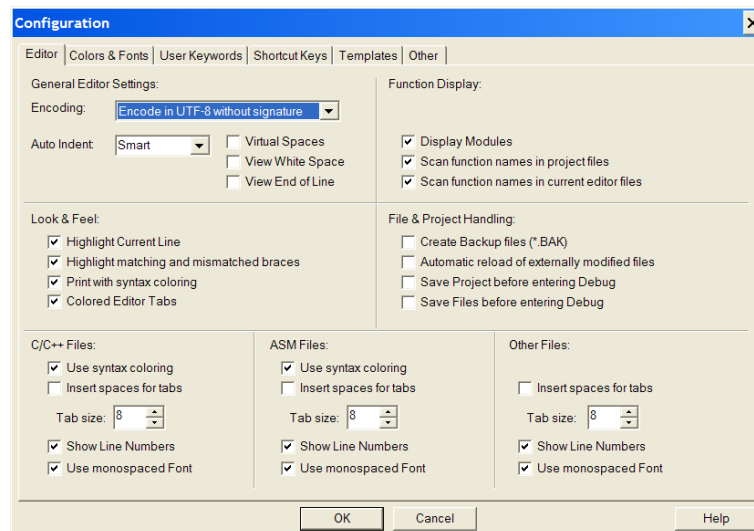


Figure 8.1 Keil µVision Editor tab of the configuration dialogue

8.2 General Comment Guidelines

Every member is to be documented in one of the following manners:

```
/**
 * <brief>
 */
<member>

/**
 * <brief>
 *
 * <description>
 */
<member>
```

- <brief>
 - A short, single sentence description of the member
- <description>
 - A detailed description of the member

8.2.1 List Formatting

Descriptions may contain syntactical sugar such as lists:

```
This is a list:
- List entry 0
- List entry 1 is a little wider than 80 characters and thus needs to cover
  multiple lines
  - List entry 1.0
- List entry 2
```

List entries are started with a dash. Every sublevel is indented by 1 tab per level. In multiple line entries the successive lines are indented 2 additional spaces to align them with the previous line.

Unless it is a keyword the first word of a list entry needs to be a capital letter. List entries do not end with a full stop.

8.2.2 Tables

The syntax for tables is:

```
| Heading 0      | Heading 1      |
|-----|-----|
| Row 0, col 0 | Row 0, col 1 |
| Row 1, col 0 | Row 1, col 1 |
```

The resulting table looks like this:

Heading 0	Heading 1
Row 0, col 0	Row 0, col 1
Row 1, col 0	Row 1, col 1

Use a colons in the separator row to align columns:

```
| Heading Left | Heading Centre | Heading Right |
|:-----:|:-----:|:-----:|
| Left      | Centre      | Right      |
```

Heading Left	Heading Centre	Heading Right
Left	Centre	Right

Note

Take care to obey the [Code/Comment Indentation and Formatting](#) guidelines, using the wrong tab width looks especially disturbing in the plain text version of a table.

8.2.3 Inline Comments

Inline comments not intended to appear in the documentation can take one of the following shapes.

Compact:

```
/* <comment> */
```

Multiline compact:

```
/* <comment line 0>
 * <comment line 1> */
```

Significant:

```
/*
 * <comment>
 */
```

8.3 Function Documentation

Every parameter of a member function and the return value if present need JavaDoc style `@param` and `@return` documentations in their descriptions:

```
/**
 * <brief>
 *
 * <description>
 *
 * @param <parameter>
 *       <parameter description>
 * @return
 *       <return value description>
 */
```

8.3.1 Return Values

Use `@retval` to document return values with logical instead of numerical meanings:

```
@retval 0
    The operation failed
@retval 1
    The operation succeeded
```

The resulting documentation takes the following appearance:

Return values

0	The operation failed
1	The operation succeeded

8.3.2 Public and Private Functions

The documentation to public functions belongs into the `.h` file. All functions that have a prototype in the header file are considered public, private functions are those, which are only used internally in the `.c` file.

If a function is public additional documentation may be placed in the `.c` file, it will only show up in the developers' manual.

Inline comments within functions may appear in JavaDoc style, in that case they are also appended to the function documentation of the developers' manual.

Private functions should be marked with `“` at the end of their documentation block.

8.4 Grouping Documentation

In some cases a set of documented members belong together, such as a set of defines for a certain function parameter. In such a cases the members can be grouped:


```

/**
 * \defgroup TRI_STATE Tri-State States
 *
 * Defines in this group represent one of the tri-state states
 *
 * @{
 */

/**
 * Tri state off.
 */
#define TRI_STATE_OFF    0

/**
 * Tri state on.
 */
#define TRI_STATE_ON     1

/**
 * Tri state high impedance.
 */
#define TRI_STATE_Z      2

/**
 * @}
 */

```

Groups are listed in the Modules chapter.

8.5 File Naming and Documentation

Three different file type suffixes are used in the construction of these libraries, `.c` for C files, `.h` for header files and `.isr` for headers that only contain ISR prototypes.

The following naming conventions exist for each file:

`hsk_<category>/hsk_<name>.<suffix>`

- `<category>`
 - The library category, often identical to name, but not necessarily so
- `<name>`
 - The name of the library
- `<suffix>`
 - The file type suffix

8.5.1 Headers

Be greedy. Only members, which are required to use a library should be listed in header files.

Header files in this project should avoid including other headers. There are two exceptions to that rule, headers containing a define to generate code might have to include headers for the generated code and headers. The second exception are [ISR Headers](#).

Every header file begins with a JavaDoc style comment:

```
/** \file
 * HSK <brief> headers
 *
 * <description>
 *
 * @author <author tag>
 */
```

- <brief>
 - A descriptive title such as "Analog Digital Conversion" this description appears in the file list, nouns, verbs and adjectives in the brief should start with capital letters
- <description>
 - A text containing all the necessary information to use the provided functions
- <author tag>
 - A short author tag from [Authors](#)
- <iso date>
 - The ISO 8601 date (YYYY-MM-DD) of the last edit

The next block contains the traditional header opening:

```
#ifndef _<FILE>_
#define _<FILE>_
```

- <FILE>
 - The file name with the following translation '[:lower:]' '[:upper:]_', e.g. [hsk_isr.h](#) becomes HSK_ISR_H

The `#ifndef` block is closed at the end of the header file with:

```
#endif /* _<FILE>_ */
```

Prototypes etc. belong within the block.

8.5.2 C Files

Like header files every C file starts with a JavaDoc style comment:

```
/** \file
 * HSK <brief> implementation
 *
 * <description>
 *
 * @author <author tag>
 */
```

- <brief>
 - Should be the same title as in the header file
- <description>

- Instead of how to use the library this should make mention of all things of interest, when working on the implementation
- `<author tag>`
 - A short author tag from [Authors](#)

The first include in a C file is the Infineon/XC878.h header, followed by the own header file. The next (optional) include block contains all the required C library headers. The final include block includes the headers of other libraries. The following example is from [hsk_adc.c](#):

```
#include <Infineon/XC878.h>

#include "hsk_adc.h"

#include <string.h> /* memset() */

#include "../hsk_isr/hsk_isr.h"
```

Comments for public members of a C file do not need to be copied from the header.

8.5.3 ISR Headers

The ISR headers exist solely for an oddity of SDCC. All interrupts must be visible from the context of the [main\(\)](#) function.

Every implementation providing an interrupt has to provide a .isr file. The file should just contain a very plain list of prototypes, the following examples is from [hsk_timer01.isr](#):

```
#ifndef _HSK_TIMER01_ISR_
#define _HSK_TIMER01_ISR_
void ISR_hsk_timer0(void) interrupt 1 using 1;
void ISR_hsk_timer1(void) interrupt 3 using 1;
#endif /* _HSK_TIMER01_ISR_ */
```

The .isr file should be included from the header file of the library providing it as well as from the header files of all libraries using that library.

The file should be included in the following manner:

```
/*
 * ISR prototypes for SDCC.
 */
#ifdef SDCC
#include "hsk_timer01.isr"
#endif /* SDCC */
```

8.6 Member Naming Conventions

All members except defines and statics have the same structure of context prefix. Contexts can be nested, each level of context is separated by an underscore. Member names following the underscore separated context are camel case. The root context is always the library, subcontexts need to have a central concept or data structure that defines them.

Defines are always specified in capitals. Thus all separation in the names of defines is done by underscore. Public defines have the library name without

Statics are considered private within the context and thus do not require a prefix.

HSK as a prefix.

8.6.1 Public Example

This subsection explains the naming conventions in public scope (i.e. in a header file) using the example of the `hsk_can` library.

Public defines are provided to interpret return values or to specify possible parameters:

```
#define CAN_ERROR            0xff
#define CAN0_IO_P10_P11    0
#define CAN1_IO_P01_P02    4
```

Typedefs give primitive data types a meaningful name for use in a certain context. This library has functions that work on CAN nodes and functions that work on message objects:

```
typedef ubyte hsk_can_node;
typedef ubyte hsk_can_msg;
```

The CAN nodes are the central structure of the library. Functions in the `hsk_can` context always take a node as the first parameter:

```
void hsk_can_init(const ubyte pins, const ulong __xdata baud);
void hsk_can_enable(const hsk_can_node node);
```

Other functions work around the concepts of messages and message data, which are represented in their context prefixes:

```
void hsk_can_msg_getData(const hsk_can_msg msg,
                        ubyte * const msgdata);
void hsk_can_data_setSignal(ubyte * const msg, const bool motorola,
                           const bool sign, const ubyte bitPos,
                           const char bitCount, const ulong idata value);
```

8.6.2 Defines

In the private context only a few rules apply to naming defines.

If defines are named after registers or bits from the μ C manual, their original spelling should be preserved, even if it means including non-capital letters:

```
#define NBTRx                0x0084
```

Apart from that special naming conventions for defines only apply to register bits. Every register bit definition is prefixed by `BIT`. Bit fields also specify a count:

```
#define BIT_RXSEL            0
#define CNT_RXSEL            3
```

Chapter 9

Authors

Authors use short tags in the code, this is the complete list of authors and the aliases they use.

Author

kami

Dominic Fandrey `dominic.fandrey@highspeed-karlsruhe.de`

kamikaze `kamikaze@bsdforen.de`

Head of Electronics Development season 2010/2011, 2011/2012

Chapter 10

Deprecated List

Global [CAN_ENDIAN_INTEL](#)

In favour of shorter and cleaner code the [hsk_can_data_getSignal\(\)](#) and [hsk_can_data_setSignal\(\)](#) functions were switched to using boolean (motorola positive) logic

Global [CAN_ENDIAN_MOTOROLA](#)

In favour of shorter and cleaner code the [hsk_can_data_getSignal\(\)](#) and [hsk_can_data_setSignal\(\)](#) functions were switched to using boolean (motorola positive) logic

Global [hsk_adc_open](#)

Use [hsk_adc_open10\(\)](#) or [hsk_adc_open8\(\)](#) as appropriate

Global [hsk_adc_warmup](#)

Use [hsk_adc_warmup10\(\)](#)

Chapter 11

Module Index

11.1 Modules

Here is a list of all modules:

CAN Node Status Fields	49
External Interrupt Channels	52
External Interrupt Triggers	54
External Interrupt Input Ports	55
Input Port Access	59
Output Port Access	61
I/O Port Pull-Up/-Down Setup	64
Variable Access	66
Pulse Width Detection Units	68
Pulse Width Times	69
Pulse Frequencies	70
Pulse Duty Times	71
SSC I/O Ports	73

Chapter 12

Data Structure Index

12.1 Data Structures

Here are the data structures with brief descriptions:

hsk_flash_struct	This struct is a template for data that can be written to the D-Flash	75
hsk_isr14_callback	Shared non-maskable interrupt routine	77
hsk_isr5_callback	Shared interrupt 5 routine	79
hsk_isr6_callback	Shared interrupt 6 routine	81
hsk_isr8_callback	Shared interrupt 8 routine	83
hsk_isr9_callback	Shared interrupt 9 routine	86

Chapter 13

File Index

13.1 File List

Here is a list of all documented files with brief descriptions:

config.h	Configuration for the Infineon XC800 Starter Kit	89
main.c	Simple test file that is not linked into the library	304
hsk_adc/hsk_adc.c	HSK Analog Digital Conversion implementation	91
hsk_adc/hsk_adc.h	HSK Analog Digital Conversion headers	101
hsk_boot/hsk_boot.c	HSK Boot implementation	107
hsk_boot/hsk_boot.h	HSK Boot headers	116
hsk_can/hsk_can.c	HSK Controller Area Network implementation	118
hsk_can/hsk_can.h	HSK Controller Area Network headers	157
hsk_ex/hsk_ex.c	HSK External Interrupt Routing implementation	176
hsk_ex/hsk_ex.h	HSK External Interrupt Routing headers	182
hsk_filter/hsk_filter.h	HSK Filter generator	186
hsk_flash/hsk_flash.c	HSK Flash Facility implementation	187
hsk_flash/hsk_flash.h	HSK Flash Facility headers	202
hsk_icm7228/hsk_icm7228.c	HSK ICM7228 8-Digit LED Display Decoder Driver implementation	207
hsk_icm7228/hsk_icm7228.h	HSK ICM7228 8-Digit LED Display Decoder Driver generator	211
hsk_io/hsk_io.h	HSK I/O headers	215
hsk_isr/hsk_isr.c	HSK Shared Interrupt Service Routine implementation	217
hsk_isr/hsk_isr.h	HSK Shared Interrupt Service Routine headers	228

hsk_pwc/hsk_pwc.c	
HSK Pulse Width Counter implementation	231
hsk_pwc/hsk_pwc.h	
HSK Pulse Width Counter headers	249
hsk_pwm/hsk_pwm.c	
HSK Pulse Width Modulation implementation	260
hsk_pwm/hsk_pwm.h	
HSK Pulse Width Modulation headers	268
hsk_ssc/hsk_ssc.c	
HSK Synchronous Serial Interface implementation	278
hsk_ssc/hsk_ssc.h	
HSK Synchronous Serial Interface headers	285
hsk_timers/hsk_timer01.c	
HSK Timer 0/1 implementation	290
hsk_timers/hsk_timer01.h	
HSK Timer 0/1 headers	296
hsk_wdt/hsk_wdt.c	
HSK Watchdog Timer implementation	299
hsk_wdt/hsk_wdt.h	
HSK Watchdog Timer headers	302

Chapter 14

Module Documentation

14.1 CAN Node Status Fields

This group of defines specifies status fields that can be queried from [hsk_can_status\(\)](#).

Macros

- `#define CAN_STATUS_LEC 0`
The Last Error Code field provides the error triggered by the last message on the bus.
- `#define CAN_STATUS_TXOK 1`
Message Transmitted Successfully.
- `#define CAN_STATUS_RXOK 2`
Message Received Successfully.
- `#define CAN_STATUS_ALERT 3`
Alert Warning.
- `#define CAN_STATUS_EWRN 4`
Error Warning Status.
- `#define CAN_STATUS_BOFF 5`
Bus-off Status.

14.1.1 Detailed Description

This group of defines specifies status fields that can be queried from [hsk_can_status\(\)](#).

14.1.2 Macro Definition Documentation

14.1.2.1 CAN_STATUS_ALERT

```
#define CAN_STATUS_ALERT 3
```

Alert Warning.

Return values

0	No warnings
1	One of the following error conditions applies: CAN_STATUS_EWRN ; CAN_STATUS_BOFF

14.1.2.2 CAN_STATUS_BOFF

```
#define CAN_STATUS_BOFF 5
```

Bus-off Status.

Return values

0	The bus is not off
1	The bus is turned off due to an error counter exceeding 256

14.1.2.3 CAN_STATUS_EWRN

```
#define CAN_STATUS_EWRN 4
```

Error Warning Status.

Return values

0	No error warnings exceeded
1	An error counter has exceeded the warning level of 96

14.1.2.4 CAN_STATUS_LEC

```
#define CAN_STATUS_LEC 0
```

The Last Error Code field provides the error triggered by the last message on the bus.

For details check table 16-8 from the User Manual 1.1.

Return values

0	No Error
1	Stuff Error, 5 consecutive bits of the same value are stuffed, this error is triggered when the stuff bit is missing
2	Form Error, the frame format was violated
3	Ack Error, the message was not acknowledged, maybe nobody else is on the bus
4	Bit1 Error, a recessive (1) bit was sent out of sync
5	Bit0 Error, a recessive (1) bit won against a dominant (0) bit
6	CRC Error, wrong checksum for a received message

14.1.2.5 CAN_STATUS_RXOK

```
#define CAN_STATUS_RXOK 2
```

Message Received Successfully.

Return values

0	No successful receptions since the last time this field was queried
1	A message was received successfully

14.1.2.6 CAN_STATUS_TXOK

```
#define CAN_STATUS_TXOK 1
```

Message Transmitted Successfully.

Return values

0	No successful transmission since TXOK was queried last time
1	A message was transmitted and acknowledged successfully

14.2 External Interrupt Channels

This group consists of defines representing external interrupt channels.

Macros

- `#define EX_EXINT0 0`
External interrupt channel EXINT0.
- `#define EX_EXINT1 1`
External interrupt channel EXINT1.
- `#define EX_EXINT2 2`
External interrupt channel EXINT2.
- `#define EX_EXINT3 3`
External interrupt channel EXINT3.
- `#define EX_EXINT4 4`
External interrupt channel EXINT4.
- `#define EX_EXINT5 5`
External interrupt channel EXINT5.
- `#define EX_EXINT6 6`
External interrupt channel EXINT6.

14.2.1 Detailed Description

This group consists of defines representing external interrupt channels.

14.2.2 Macro Definition Documentation

14.2.2.1 EX_EXINT0

```
#define EX_EXINT0 0
```

External interrupt channel EXINT0.

Mask with EA, disable with EX0.

14.2.2.2 EX_EXINT1

```
#define EX_EXINT1 1
```

External interrupt channel EXINT1.

Mask with EA, disable with EX1.

14.2.2.3 EX_EXINT2

```
#define EX_EXINT2 2
```

External interrupt channel EXINT2.

Mask with EX2.

14.2.2.4 EX_EXINT3

```
#define EX_EXINT3 3
```

External interrupt channel EXINT3.

Mask with EXM.

14.2.2.5 EX_EXINT4

```
#define EX_EXINT4 4
```

External interrupt channel EXINT4.

Mask with EXM.

14.2.2.6 EX_EXINT5

```
#define EX_EXINT5 5
```

External interrupt channel EXINT5.

Mask with EXM.

14.2.2.7 EX_EXINT6

```
#define EX_EXINT6 6
```

External interrupt channel EXINT6.

Mask with EXM.

14.3 External Interrupt Triggers

This group contains defines representing the different edge triggers.

Macros

- `#define EX_EDGE_DISABLE 3`
Deactivate external interrupt.
- `#define EX_EDGE_RISING 0`
Trigger interrupt on rising edge.
- `#define EX_EDGE_FALLING 1`
Trigger interrupt on falling edge.
- `#define EX_EDGE_BOTH 2`
Trigger interrupt on both edges.

14.3.1 Detailed Description

This group contains defines representing the different edge triggers.

14.3.2 Macro Definition Documentation

14.3.2.1 EX_EDGE_BOTH

```
#define EX_EDGE_BOTH 2
```

Trigger interrupt on both edges.

14.3.2.2 EX_EDGE_DISABLE

```
#define EX_EDGE_DISABLE 3
```

Deactivate external interrupt.

14.3.2.3 EX_EDGE_FALLING

```
#define EX_EDGE_FALLING 1
```

Trigger interrupt on falling edge.

14.3.2.4 EX_EDGE_RISING

```
#define EX_EDGE_RISING 0
```

Trigger interrupt on rising edge.

14.4 External Interrupt Input Ports

Each define of this group represents an external interrupt port configuration.

Macros

- `#define EX_EXINT0_P05 0`
External interrupt EXINT0 input port P0.5.
- `#define EX_EXINT3_P11 1`
External interrupt EXINT3 input port P1.1.
- `#define EX_EXINT0_P14 2`
External interrupt EXINT0 input port P1.4.
- `#define EX_EXINT5_P15 3`
External interrupt EXINT5 input port P1.5.
- `#define EX_EXINT6_P16 4`
External interrupt EXINT6 input port P1.6.
- `#define EX_EXINT3_P30 5`
External interrupt EXINT3 input port P3.0.
- `#define EX_EXINT4_P32 6`
External interrupt EXINT4 input port P3.2.
- `#define EX_EXINT5_P33 7`
External interrupt EXINT5 input port P3.3.
- `#define EX_EXINT6_P34 8`
External interrupt EXINT6 input port P3.4.
- `#define EX_EXINT4_P37 9`
External interrupt EXINT4 input port P3.7.
- `#define EX_EXINT3_P40 10`
External interrupt EXINT3 input port P4.0.
- `#define EX_EXINT4_P41 11`
External interrupt EXINT4 input port P4.1.
- `#define EX_EXINT6_P42 12`
External interrupt EXINT6 input port P4.2.
- `#define EX_EXINT5_P44 13`
External interrupt EXINT5 input port P4.4.
- `#define EX_EXINT6_P45 14`
External interrupt EXINT6 input port P4.5.
- `#define EX_EXINT1_P50 15`
External interrupt EXINT1 input port P5.0.
- `#define EX_EXINT2_P51 16`
External interrupt EXINT2 input port P5.1.
- `#define EX_EXINT5_P52 17`
External interrupt EXINT5 input port P5.2.
- `#define EX_EXINT1_P53 18`
External interrupt EXINT1 input port P5.3.
- `#define EX_EXINT2_P54 19`
External interrupt EXINT2 input port P5.4.
- `#define EX_EXINT3_P55 20`
External interrupt EXINT3 input port P5.5.
- `#define EX_EXINT4_P56 21`
External interrupt EXINT4 input port P5.6.
- `#define EX_EXINT6_P57 22`
External interrupt EXINT6 input port P5.7.

14.4.1 Detailed Description

Each define of this group represents an external interrupt port configuration.

14.4.2 Macro Definition Documentation

14.4.2.1 EX_EXINT0_P05

```
#define EX_EXINT0_P05 0
```

External interrupt EXINT0 input port P0.5.

14.4.2.2 EX_EXINT0_P14

```
#define EX_EXINT0_P14 2
```

External interrupt EXINT0 input port P1.4.

14.4.2.3 EX_EXINT1_P50

```
#define EX_EXINT1_P50 15
```

External interrupt EXINT1 input port P5.0.

14.4.2.4 EX_EXINT1_P53

```
#define EX_EXINT1_P53 18
```

External interrupt EXINT1 input port P5.3.

14.4.2.5 EX_EXINT2_P51

```
#define EX_EXINT2_P51 16
```

External interrupt EXINT2 input port P5.1.

14.4.2.6 EX_EXINT2_P54

```
#define EX_EXINT2_P54 19
```

External interrupt EXINT2 input port P5.4.

14.4.2.7 EX_EXINT3_P11

```
#define EX_EXINT3_P11 1
```

External interrupt EXINT3 input port P1.1.

14.4.2.8 EX_EXINT3_P30

```
#define EX_EXINT3_P30 5
```

External interrupt EXINT3 input port P3.0.

14.4.2.9 EX_EXINT3_P40

```
#define EX_EXINT3_P40 10
```

External interrupt EXINT3 input port P4.0.

14.4.2.10 EX_EXINT3_P55

```
#define EX_EXINT3_P55 20
```

External interrupt EXINT3 input port P5.5.

14.4.2.11 EX_EXINT4_P32

```
#define EX_EXINT4_P32 6
```

External interrupt EXINT4 input port P3.2.

14.4.2.12 EX_EXINT4_P37

```
#define EX_EXINT4_P37 9
```

External interrupt EXINT4 input port P3.7.

14.4.2.13 EX_EXINT4_P41

```
#define EX_EXINT4_P41 11
```

External interrupt EXINT4 input port P4.1.

14.4.2.14 EX_EXINT4_P56

```
#define EX_EXINT4_P56 21
```

External interrupt EXINT4 input port P5.6.

14.4.2.15 EX_EXINT5_P15

```
#define EX_EXINT5_P15 3
```

External interrupt EXINT5 input port P1.5.

14.4.2.16 EX_EXINT5_P33

```
#define EX_EXINT5_P33 7
```

External interrupt EXINT5 input port P3.3.

14.4.2.17 EX_EXINT5_P44

```
#define EX_EXINT5_P44 13
```

External interrupt EXINT5 input port P4.4.

14.4.2.18 EX_EXINT5_P52

```
#define EX_EXINT5_P52 17
```

External interrupt EXINT5 input port P5.2.

14.4.2.19 EX_EXINT6_P16

```
#define EX_EXINT6_P16 4
```

External interrupt EXINT6 input port P1.6.

14.4.2.20 EX_EXINT6_P34

```
#define EX_EXINT6_P34 8
```

External interrupt EXINT6 input port P3.4.

14.4.2.21 EX_EXINT6_P42

```
#define EX_EXINT6_P42 12
```

External interrupt EXINT6 input port P4.2.

14.4.2.22 EX_EXINT6_P45

```
#define EX_EXINT6_P45 14
```

External interrupt EXINT6 input port P4.5.

14.4.2.23 EX_EXINT6_P57

```
#define EX_EXINT6_P57 22
```

External interrupt EXINT6 input port P5.7.

14.5 Input Port Access

This group contains defines and macros to initialize port pins as inputs and read them.

Macros

- `#define IO_PORT_IN_INIT(port, pins)`
Initializes a set of port pins as inputs.
- `#define IO_PORT_ON_GND 0`
Bit mask to set the logical 1 to GND level for all selected pins.
- `#define IO_PORT_ON_HIGH 0xff`
Bit mask to set the logical 1 to high level for all selected pins.
- `#define IO_PORT_GET(port, pins, on)`
Evaluates to a bit mask of logical pin states of a port.

14.5.1 Detailed Description

This group contains defines and macros to initialize port pins as inputs and read them.

14.5.2 Macro Definition Documentation

14.5.2.1 IO_PORT_GET

```
#define IO_PORT_GET(  
    port,  
    pins,  
    on )
```

Value:

```
( \  
    (port##_DATA ^ ~(on)) & (pins) \  
)
```

Evaluates to a bit mask of logical pin states of a port.

Note

Can also be used for [Output Port Access](#)

Warning

Expects port page 0 and RMAP 0, take care in ISRs

Parameters

<i>port</i>	The parallel port to access
<i>pins</i>	A bit mask of the pins to select
<i>on</i>	A bit mask of pins that defines the states which represent on

14.5.2.2 IO_PORT_IN_INIT

```
#define IO_PORT_IN_INIT(  
    port,  
    pins )
```

Value:

```
{ \n    port##_DIR &= ~(pins); \n}
```

Initializes a set of port pins as inputs.

Warning

Expects port page 0 and RMAP 0, take care in ISRs

Parameters

<i>port</i>	The parallel port to configure
<i>pins</i>	A bit mask of the pins to select

14.5.2.3 IO_PORT_ON_GND

```
#define IO_PORT_ON_GND 0
```

Bit mask to set the logical 1 to GND level for all selected pins.

Note

Can also be used for [Output Port Access](#)

14.5.2.4 IO_PORT_ON_HIGH

```
#define IO_PORT_ON_HIGH 0xff
```

Bit mask to set the logical 1 to high level for all selected pins.

Note

Can also be used for [Output Port Access](#)

14.6 Output Port Access

This group contains macros and defines to initialize port pins for output and safely set output states.

Macros

- `#define IO_PORT_STRENGTH_WEAK 0`
Bit mask to set weak drive strength for all selected pins.
- `#define IO_PORT_STRENGTH_STRONG 0xff`
Bit mask to set strong drive strength for all selected pins.
- `#define IO_PORT_DRAIN_DISABLE 0`
Bit mask to disable drain mode for all selected pins.
- `#define IO_PORT_DRAIN_ENABLE 0xff`
Bit mask to enable drain mode for all selected pins.
- `#define IO_PORT_OUT_INIT(port, pins, strength, drain, on, set)`
Initializes a set of port pins as outputs.
- `#define IO_PORT_OUT_SET(port, pins, on, set)`
Set a set of output port pins.

14.6.1 Detailed Description

This group contains macros and defines to initialize port pins for output and safely set output states.

14.6.2 Macro Definition Documentation

14.6.2.1 IO_PORT_DRAIN_DISABLE

```
#define IO_PORT_DRAIN_DISABLE 0
```

Bit mask to disable drain mode for all selected pins.

14.6.2.2 IO_PORT_DRAIN_ENABLE

```
#define IO_PORT_DRAIN_ENABLE 0xff
```

Bit mask to enable drain mode for all selected pins.

14.6.2.3 IO_PORT_OUT_INIT

```
#define IO_PORT_OUT_INIT(
    port,
    pins,
    strength,
    drain,
    on,
    set )
```

Value:

```
{ \
    port##_DIR |= pins; \
    SFR_PAGE(_pp3, noSST); \
    port##_OD &= (drain) | ~(pins); \
    port##_OD |= (drain) & (pins); \
    port##_DS &= (strength) | ~(pins); \
    port##_DS |= (strength) & (pins); \
    SFR_PAGE(_pp0, noSST); \
    port##_DATA &= ((set) ^ ~(on)) | ~(pins); \
    port##_DATA |= ((set) ^ ~(on)) & (pins); \
}
```

Initializes a set of port pins as outputs.

Warning

Expects port page 0 and RMAP 0, take care in ISRs

Parameters

<i>port</i>	The parallel port to configure
<i>pins</i>	A bit mask of the pins to select
<i>strength</i>	A bit mask of pins with strong drive strength
<i>drain</i>	A bit mask of pins that only drive GND
<i>on</i>	A bit mask of pins that defines the states which represent on

See also

[IO_PORT_ON_GND](#)
[IO_PORT_ON_HIGH](#)

Parameters

<i>set</i>	Initial logical values for the defined outputs
------------	--

14.6.2.4 IO_PORT_OUT_SET

```
#define IO_PORT_OUT_SET(
    port,
    pins,
```

```

    on,
    set )

```

Value:

```

{ \
    port##_DATA &= ((set) ^ ~(on)) | ~(pins); \
    port##_DATA |= ((set) ^ ~(on)) & (pins); \
}

```

Set a set of output port pins.

Warning

Expects port page 0 and RMAP 0, take care in ISRs

Parameters

<i>port</i>	The parallel port to set
<i>pins</i>	A bit mask of the pins to select
<i>on</i>	A bit mask of pins that defines the states which represent on

See also

[IO_PORT_ON_GND](#)
[IO_PORT_ON_HIGH](#)

Parameters

<i>set</i>	Set logical values for the defined outputs
------------	--

14.6.2.5 IO_PORT_STRENGTH_STRONG

```
#define IO_PORT_STRENGTH_STRONG 0xff
```

Bit mask to set strong drive strength for all selected pins.

14.6.2.6 IO_PORT_STRENGTH_WEAK

```
#define IO_PORT_STRENGTH_WEAK 0
```

Bit mask to set weak drive strength for all selected pins.

14.7 I/O Port Pull-Up/-Down Setup

This group contains macros and defines to initialize the pull-up/-down devices of port pins.

Macros

- `#define IO_PORT_PULL_DISABLE 0`
Bit mask to disable pull up/down for all selected pins.
- `#define IO_PORT_PULL_ENABLE 0xff`
Bit mask to enable pull up/down for all selected pins.
- `#define IO_PORT_PULL_DOWN 0`
Bit mask to select pull down for all selected pins.
- `#define IO_PORT_PULL_UP 0xff`
Bit mask to select pull up for all selected pins.
- `#define IO_PORT_PULL_INIT(port, pins, pull, dir)`
Sets the pull-up/-down properties of port pins.

14.7.1 Detailed Description

This group contains macros and defines to initialize the pull-up/-down devices of port pins.

14.7.2 Macro Definition Documentation

14.7.2.1 IO_PORT_PULL_DISABLE

```
#define IO_PORT_PULL_DISABLE 0
```

Bit mask to disable pull up/down for all selected pins.

14.7.2.2 IO_PORT_PULL_DOWN

```
#define IO_PORT_PULL_DOWN 0
```

Bit mask to select pull down for all selected pins.

14.7.2.3 IO_PORT_PULL_ENABLE

```
#define IO_PORT_PULL_ENABLE 0xff
```

Bit mask to enable pull up/down for all selected pins.

14.7.2.4 IO_PORT_PULL_INIT

```
#define IO_PORT_PULL_INIT(
    port,
    pins,
    pull,
    dir )
```

Value:

```
{ \
    SFR_PAGE(_pp1, noSST); \
    port##_PUDSEL &= (dir) | ~(pins); \
    port##_PUDSEL |= (dir) & (pins); \
    port##_PU DEN &= (pull) | ~(pins); \
    port##_PU DEN |= (pull) & (pins); \
    SFR_PAGE(_pp0, noSST); \
}
```

Sets the pull-up/-down properties of port pins.

Warning

Expects port page 0 and RMAP 0, take care in ISRs

Parameters

<i>port</i>	The parallel port to configure
<i>pins</i>	A bit mask of the pins to select
<i>pull</i>	A bit mask of pins to activate the internal pull up/down device for
<i>dir</i>	A bit mask of pins to set the pull direction

14.7.2.5 IO_PORT_PULL_UP

```
#define IO_PORT_PULL_UP 0xff
```

Bit mask to select pull up for all selected pins.

14.8 Variable Access

This group specifies macros to access bits of a variable.

Macros

- `#define IO_VAR_SET(var, bits, on, set)`
Set a set of variable bits.
- `#define IO_VAR_GET(var, bits, on)`
Evaluates to a bit mask of logical states of a variable.

14.8.1 Detailed Description

This group specifies macros to access bits of a variable.

Their value lies in the separation of encoded `on` state and logical `on` (1), as well as the safe bit masking.

14.8.2 Macro Definition Documentation

14.8.2.1 IO_VAR_GET

```
#define IO_VAR_GET(  
    var,  
    bits,  
    on )
```

Value:

```
( \  
    ((var) ^ ~(on)) & (bits) \  
)
```

Evaluates to a bit mask of logical states of a variable.

Parameters

<i>var</i>	The variable to access
<i>bits</i>	A bit mask of the bits to select
<i>on</i>	A bit mask that defines the states which represent true

14.8.2.2 IO_VAR_SET

```
#define IO_VAR_SET(  
    var,  
    bits,
```



```
on,  
set )
```

Value:

```
{\  
    (var) &= ((set) ^ ~(on)) | ~(bits); \  
    (var) |= ((set) ^ ~(on)) & (bits); \  
}
```

Set a set of variable bits.

Parameters

<i>var</i>	The variable to set
<i>bits</i>	A bit mask of the bits to select
<i>on</i>	A bit mask that defines the states which represent true
<i>set</i>	Set logical values for the defined bits

14.9 Pulse Width Detection Units

This group of defines is used to select return format of [hsk_pwc_channel_getValue\(\)](#).

Modules

- [Pulse Width Times](#)

The defines are for returning average pulse width.

- [Pulse Frequencies](#)

These defines are for returning average frequencies.

- [Pulse Duty Times](#)

These defines are used for returning the duty time of the latest pulse.

Macros

- `#define PWC_UNIT_SUM_RAW 0`

*Sum of buffered pulse widths in multiples of $1/48 * 10^{-6} s$.*

14.9.1 Detailed Description

This group of defines is used to select return format of [hsk_pwc_channel_getValue\(\)](#).

14.9.2 Macro Definition Documentation

14.9.2.1 PWC_UNIT_SUM_RAW

```
#define PWC_UNIT_SUM_RAW 0
```

Sum of buffered pulse widths in multiples of $1/48 * 10^{-6} s$.

This is the sum of the buffered values, not the average.

Use this if precision is of the utmost importance.

14.10 Pulse Width Times

The defines are for returning average pulse width.

Macros

- `#define PWC_UNIT_WIDTH_RAW 1`
*Average of buffered pulse widths in multiples of $1/48 * 10^{-6} s$.*
- `#define PWC_UNIT_WIDTH_NS 2`
Average of buffered pulse widths in multiples of $10^{-9} s$.
- `#define PWC_UNIT_WIDTH_US 3`
Average of buffered pulse widths in multiples of $10^{-6} s$.
- `#define PWC_UNIT_WIDTH_MS 4`
Average of buffered pulse widths in multiples of $10^{-3} s$.

14.10.1 Detailed Description

The defines are for returning average pulse width.

14.10.2 Macro Definition Documentation

14.10.2.1 PWC_UNIT_WIDTH_MS

```
#define PWC_UNIT_WIDTH_MS 4
```

Average of buffered pulse widths in multiples of $10^{-3} s$.

14.10.2.2 PWC_UNIT_WIDTH_NS

```
#define PWC_UNIT_WIDTH_NS 2
```

Average of buffered pulse widths in multiples of $10^{-9} s$.

14.10.2.3 PWC_UNIT_WIDTH_RAW

```
#define PWC_UNIT_WIDTH_RAW 1
```

Average of buffered pulse widths in multiples of $1/48 * 10^{-6} s$.

14.10.2.4 PWC_UNIT_WIDTH_US

```
#define PWC_UNIT_WIDTH_US 3
```

Average of buffered pulse widths in multiples of $10^{-6} s$.

14.11 Pulse Frequencies

These defines are for returning average frequencies.

Macros

- `#define PWC_UNIT_FREQ_S 5`
Average frequency of buffered pulses in multiples of $1/s$.
- `#define PWC_UNIT_FREQ_M 6`
Average frequency of buffered pulses in multiples of $1/m$.
- `#define PWC_UNIT_FREQ_H 7`
Average frequency of buffered pulses in multiples of $1/h$.

14.11.1 Detailed Description

These defines are for returning average frequencies.

14.11.2 Macro Definition Documentation

14.11.2.1 PWC_UNIT_FREQ_H

```
#define PWC_UNIT_FREQ_H 7
```

Average frequency of buffered pulses in multiples of $1/h$.

To prevent overflow issues this value is always a multiple of the number of averaged values * 60.

This is just a convenience feature for quick testing, it is possible to achieve much better precision if the use case is known.

14.11.2.2 PWC_UNIT_FREQ_M

```
#define PWC_UNIT_FREQ_M 6
```

Average frequency of buffered pulses in multiples of $1/m$.

To prevent overflow issues this value is always a multiple of the number of averaged values.

14.11.2.3 PWC_UNIT_FREQ_S

```
#define PWC_UNIT_FREQ_S 5
```

Average frequency of buffered pulses in multiples of $1/s$.

14.12 Pulse Duty Times

These defines are used for returning the duty time of the latest pulse.

Macros

- `#define PWC_UNIT_DUTYH_RAW 8`
*Latest high pulse in multiples of $1/48 * 10^{-6} s$.*
- `#define PWC_UNIT_DUTYH_NS 9`
*Latest high pulse in multiples of $1 * 10^{-9} s$.*
- `#define PWC_UNIT_DUTYH_US 10`
*Latest high pulse in multiples of $1 * 10^{-6} s$.*
- `#define PWC_UNIT_DUTYH_MS 11`
*Latest high pulse in multiples of $1 * 10^{-3} s$.*
- `#define PWC_UNIT_DUTYL_RAW 12`
*Latest low pulse in multiples of $1/48 * 10^{-6} s$.*
- `#define PWC_UNIT_DUTYL_NS 13`
*Latest low pulse in multiples of $1 * 10^{-9} s$.*
- `#define PWC_UNIT_DUTYL_US 14`
*Latest low pulse in multiples of $1 * 10^{-6} s$.*
- `#define PWC_UNIT_DUTYL_MS 15`
*Latest low pulse in multiples of $1 * 10^{-3} s$.*

14.12.1 Detailed Description

These defines are used for returning the duty time of the latest pulse.

In order to use this return type, the channel buffer must hold at least 2 values. I.e. the `averageOver` argument of `hsk_pwc_port_open()` must be 2 or greater (there is no benefit to a value above 2).

To produce correct results the channel must also be in edge mode `PWC_EDGE_BOTH`.

14.12.2 Macro Definition Documentation

14.12.2.1 PWC_UNIT_DUTYH_MS

```
#define PWC_UNIT_DUTYH_MS 11
```

Latest high pulse in multiples of $1 * 10^{-3} s$.

14.12.2.2 PWC_UNIT_DUTYH_NS

```
#define PWC_UNIT_DUTYH_NS 9
```

Latest high pulse in multiples of $1 * 10^{-9} s$.

14.12.2.3 PWC_UNIT_DUTYH_RAW

```
#define PWC_UNIT_DUTYH_RAW 8
```

Latest high pulse in multiples of $1/48 * 10^{-6}s$.

14.12.2.4 PWC_UNIT_DUTYH_US

```
#define PWC_UNIT_DUTYH_US 10
```

Latest high pulse in multiples of $1 * 10^{-6}s$.

14.12.2.5 PWC_UNIT_DUTYL_MS

```
#define PWC_UNIT_DUTYL_MS 15
```

Latest low pulse in multiples of $1 * 10^{-3}s$.

14.12.2.6 PWC_UNIT_DUTYL_NS

```
#define PWC_UNIT_DUTYL_NS 13
```

Latest low pulse in multiples of $1 * 10^{-9}s$.

14.12.2.7 PWC_UNIT_DUTYL_RAW

```
#define PWC_UNIT_DUTYL_RAW 12
```

Latest low pulse in multiples of $1/48 * 10^{-6}s$.

14.12.2.8 PWC_UNIT_DUTYL_US

```
#define PWC_UNIT_DUTYL_US 14
```

Latest low pulse in multiples of $1 * 10^{-6}s$.

14.13 SSC I/O Ports

Used to create an I/O Port configuration, by unifying one of the SSC_MRST_P* with a SSC_MTSR_P* and a SSC_SCLK_P* ports.

Macros

- `#define SSC_MRST_P05 1`
Master mode RX, slave mode TX port P0.5.
- `#define SSC_MRST_P14 0`
Master mode RX, slave mode TX port P1.4.
- `#define SSC_MRST_P15 2`
Master mode RX, slave mode TX port P1.5.
- `#define SSC_MTSR_P04 (1 << 2)`
Master mode TX, slave mode RX port P0.4.
- `#define SSC_MTSR_P13 (0 << 2)`
Master mode TX, slave mode RX port P1.3.
- `#define SSC_MTSR_P14 (2 << 2)`
Master mode TX, slave mode RX port P1.4.
- `#define SSC_SCLK_P03 (1 << 4)`
Synchronous clock port P0.3.
- `#define SSC_SCLK_P12 (0 << 4)`
Synchronous clock port P1.2.
- `#define SSC_SCLK_P13 (2 << 4)`
Synchronous clock port P1.3.

14.13.1 Detailed Description

Used to create an I/O Port configuration, by unifying one of the SSC_MRST_P* with a SSC_MTSR_P* and a SSC_SCLK_P* ports.

E.g.:

```
SSC_MRST_P05 | SSC_MTSR_P4 | SSC_SCLK_P03.
```

The ports have the following functions:

Type	Master Mode	Slave Mode
MRST	RX port	TX port
MTSR	TX port	RX port
SCLK	TX clock	RX clock

14.13.2 Macro Definition Documentation

14.13.2.1 SSC_MRST_P05

```
#define SSC_MRST_P05 1
```

Master mode RX, slave mode TX port P0.5.

14.13.2.2 SSC_MRST_P14

```
#define SSC_MRST_P14 0
```

Master mode RX, slave mode TX port P1.4.

14.13.2.3 SSC_MRST_P15

```
#define SSC_MRST_P15 2
```

Master mode RX, slave mode TX port P1.5.

14.13.2.4 SSC_MTSR_P04

```
#define SSC_MTSR_P04 (1 << 2)
```

Master mode TX, slave mode RX port P0.4.

14.13.2.5 SSC_MTSR_P13

```
#define SSC_MTSR_P13 (0 << 2)
```

Master mode TX, slave mode RX port P1.3.

14.13.2.6 SSC_MTSR_P14

```
#define SSC_MTSR_P14 (2 << 2)
```

Master mode TX, slave mode RX port P1.4.

14.13.2.7 SSC_SCLK_P03

```
#define SSC_SCLK_P03 (1 << 4)
```

Synchronous clock port P0.3.

14.13.2.8 SSC_SCLK_P12

```
#define SSC_SCLK_P12 (0 << 4)
```

Synchronous clock port P1.2.

14.13.2.9 SSC_SCLK_P13

```
#define SSC_SCLK_P13 (2 << 4)
```

Synchronous clock port P1.3.

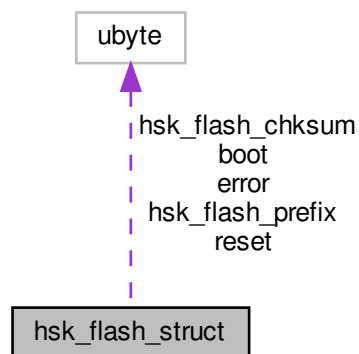
Chapter 15

Data Structure Documentation

15.1 hsk_flash_struct Struct Reference

This struct is a template for data that can be written to the D-Flash.

Collaboration diagram for hsk_flash_struct:



Data Fields

- ubyte [boot](#)
Used for boot counting.
- ubyte [reset](#)
Used for reset counting.
- ubyte [error](#)
For storing errors.

Private Attributes

- ubyte [hsk_flash_prefix](#)
For data integrity/compatibility detection.
- ubyte [hsk_flash_chksum](#)
For data integrity detection.

15.1.1 Detailed Description

This struct is a template for data that can be written to the D-Flash.

It is created by invoking the [FLASH_STRUCT_FACTORY](#) macro.

15.1.2 Field Documentation

15.1.2.1 boot

```
ubyte hsk_flash_struct::boot
```

Used for boot counting.

15.1.2.2 error

```
ubyte hsk_flash_struct::error
```

For storing errors.

Certain errors like a WDT can only be reported after a reboot.

15.1.2.3 hsk_flash_chksum

```
ubyte hsk_flash_struct::hsk_flash_chksum [private]
```

For data integrity detection.

15.1.2.4 hsk_flash_prefix

```
ubyte hsk_flash_struct::hsk_flash_prefix [private]
```

For data integrity/compatibility detection.

15.1.2.5 reset

```
ubyte hsk_flash_struct::reset
```

Used for reset counting.

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

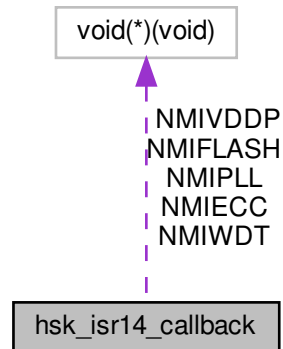
- [main.c](#)

15.2 hsk_isr14_callback Struct Reference

Shared non-maskable interrupt routine.

```
#include <hsk_isr.h>
```

Collaboration diagram for hsk_isr14_callback:



Data Fields

- void(* [NMIWDT](#))(void)
Function to be called back when the NMIWDT interrupt event is triggered.
- void(* [NMIPLL](#))(void)
Function to be called back when the NMIPLL interrupt event is triggered.
- void(* [NMIFLASH](#))(void)
Function to be called back when the NMIFLASH interrupt event is triggered.
- void(* [NMIVDDP](#))(void)
Function to be called back when the NMIVDDP interrupt event is triggered.
- void(* [NMIECC](#))(void)
Function to be called back when the NMIECC interrupt event is triggered.

15.2.1 Detailed Description

Shared non-maskable interrupt routine.

This interrupt has the following sources:

- Watchdog Timer NMI (NMIWDT)
- PLL NMI (NMIPLL)
- Flash Timer NMI (NMIFLASH)
- VDDP Prewarning NMI (NMIVDDP)
- Flash ECC NMI (NMIECC)

15.2.2 Field Documentation

15.2.2.1 NMIECC

```
void( * hsk_isr14_callback::NMIECC) (void)
```

Function to be called back when the NMIECC interrupt event is triggered.

15.2.2.2 NMIFLASH

```
void( * hsk_isr14_callback::NMIFLASH) (void)
```

Function to be called back when the NMIFLASH interrupt event is triggered.

15.2.2.3 NMIPLL

```
void( * hsk_isr14_callback::NMIPLL) (void)
```

Function to be called back when the NMIPLL interrupt event is triggered.

15.2.2.4 NMIVDDP

```
void( * hsk_isr14_callback::NMIVDDP) (void)
```

Function to be called back when the NMIVDDP interrupt event is triggered.

15.2.2.5 NMIWDT

```
void( * hsk_isr14_callback::NMIWDT) (void)
```

Function to be called back when the NMIWDT interrupt event is triggered.

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

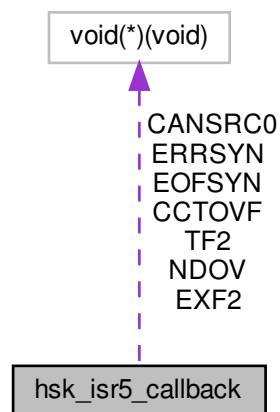
- [hsk_isr/hsk_isr.h](#)

15.3 hsk_isr5_callback Struct Reference

Shared interrupt 5 routine.

```
#include <hsk_isr.h>
```

Collaboration diagram for hsk_isr5_callback:



Data Fields

- `void(* TF2)(void)`
Function to be called back when the `TF2` interrupt event is triggered.
- `void(* EXF2)(void)`
Function to be called back when the `EXF2` interrupt event is triggered.
- `void(* CCTOVF)(void)`
Function to be called back when the `CCTOVF` interrupt event is triggered.
- `void(* NDOV)(void)`
Function to be called back when the `NDOV` interrupt event is triggered.
- `void(* EOFSYN)(void)`
Function to be called back when the `EOFSYN` interrupt event is triggered.
- `void(* ERRSYN)(void)`
Function to be called back when the `ERRSYN` interrupt event is triggered.
- `void(* CANSRC0)(void)`
Function to be called back when the `CANSRC0` interrupt event is triggered.

15.3.1 Detailed Description

Shared interrupt 5 routine.

Activate the interrupt by setting ET2 = 1.

This interrupt has the following sources:

- Timer 2 Overflow (TF2)
- Timer 2 External Event (EXF2)
- T2CCU CCT Overflow (CCTOVF)
- Normal Divider Overflow (NDOV)
- End of Syn Byte (EOFSYN)
- Syn Byte Error (ERRSYN)
- CAN Interrupt 0 (CANSRC0)

15.3.2 Field Documentation

15.3.2.1 CANSRC0

```
void( * hsk_isr5_callback::CANSRC0) (void)
```

Function to be called back when the CANSRC0 interrupt event is triggered.

15.3.2.2 CCTOVF

```
void( * hsk_isr5_callback::CCTOVF) (void)
```

Function to be called back when the CCTOVF interrupt event is triggered.

15.3.2.3 EOFSYN

```
void( * hsk_isr5_callback::EOFSYN) (void)
```

Function to be called back when the EOFSYN interrupt event is triggered.

15.3.2.4 ERRSYN

```
void( * hsk_isr5_callback::ERRSYN) (void)
```

Function to be called back when the ERRSYN interrupt event is triggered.

15.3.2.5 EXF2

```
void( * hsk_isr5_callback::EXF2) (void)
```

Function to be called back when the EXF2 interrupt event is triggered.

15.3.2.6 NDOV

```
void( * hsk_isr5_callback::NDOV) (void)
```

Function to be called back when the NDOV interrupt event is triggered.

15.3.2.7 TF2

```
void( * hsk_isr5_callback::TF2) (void)
```

Function to be called back when the TF2 interrupt event is triggered.

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

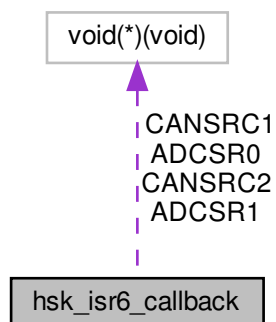
- [hsk_isr/hsk_isr.h](#)

15.4 hsk_isr6_callback Struct Reference

Shared interrupt 6 routine.

```
#include <hsk_isr.h>
```

Collaboration diagram for hsk_isr6_callback:



Data Fields

- void(* [CANSRC1](#))(void)
Function to be called back when the CANSRC1 interrupt event is triggered.
- void(* [CANSRC2](#))(void)
Function to be called back when the CANSRC2 interrupt event is triggered.
- void(* [ADCSR0](#))(void)
Function to be called back when the ADCSR0 interrupt event is triggered.
- void(* [ADCSR1](#))(void)
Function to be called back when the ADCSR1 interrupt event is triggered.

15.4.1 Detailed Description

Shared interrupt 6 routine.

Activate the interrupt by setting EADC = 1.

This interrupt has the following sources:

- CANSRC1
- CANSRC2
- ADCSR0
- ADCSR1

15.4.2 Field Documentation

15.4.2.1 ADCSR0

```
void( * hsk_isr6_callback::ADCSR0) (void)
```

Function to be called back when the ADCSR0 interrupt event is triggered.

15.4.2.2 ADCSR1

```
void( * hsk_isr6_callback::ADCSR1) (void)
```

Function to be called back when the ADCSR1 interrupt event is triggered.

15.4.2.3 CANSRC1

```
void( * hsk_isr6_callback::CANSRC1) (void)
```

Function to be called back when the CANSRC1 interrupt event is triggered.

15.4.2.4 CANSRC2

```
void( * hsk_isr6_callback::CANSRC2) (void)
```

Function to be called back when the CANSRC2 interrupt event is triggered.

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

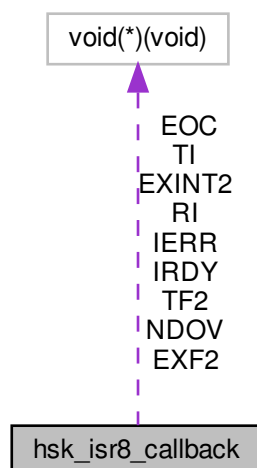
- [hsk_isr/hsk_isr.h](#)

15.5 hsk_isr8_callback Struct Reference

Shared interrupt 8 routine.

```
#include <hsk_isr.h>
```

Collaboration diagram for hsk_isr8_callback:



Data Fields

- `void(* EXINT2)(void)`
Function to be called back when the EXINT2 interrupt event is triggered.
- `void(* RI)(void)`
Function to be called back when the RI interrupt event is triggered.
- `void(* TI)(void)`
Function to be called back when the TI interrupt event is triggered.
- `void(* TF2)(void)`
Function to be called back when the TF2 interrupt event is triggered.
- `void(* EXF2)(void)`

Function to be called back when the EXF2 interrupt event is triggered.

- void(* [NDOV](#))(void)

Function to be called back when the NDOV interrupt event is triggered.

- void(* [EOC](#))(void)

Function to be called back when the EOC interrupt event is triggered.

- void(* [IRDY](#))(void)

Function to be called back when the IRDY interrupt event is triggered.

- void(* [IERR](#))(void)

Function to be called back when the IERR interrupt event is triggered.

15.5.1 Detailed Description

Shared interrupt 8 routine.

Activate the interrupt by setting EX2 = 1.

This interrupt has the following sources:

- External Interrupt 2 (EXINT2)
- UART1 (RI)
- UART1 (TI)
- Timer 21 Overflow (TF2)
- T21EX (EXF2)
- UART1 Fractional Divider (Normal Divider Overflow) (NDOV)
- CORDIC (EOC)
- MDU Result Ready (IRDY)
- MDU Error (IERR)

15.5.2 Field Documentation

15.5.2.1 EOC

```
void( * hsk_isr8_callback::EOC) (void)
```

Function to be called back when the EOC interrupt event is triggered.

15.5.2.2 EXF2

```
void( * hsk_isr8_callback::EXF2) (void)
```

Function to be called back when the EXF2 interrupt event is triggered.

15.5.2.3 EXINT2

```
void( * hsk_isr8_callback::EXINT2) (void)
```

Function to be called back when the EXINT2 interrupt event is triggered.

15.5.2.4 IERR

```
void( * hsk_isr8_callback::IERR) (void)
```

Function to be called back when the IERR interrupt event is triggered.

15.5.2.5 IRDY

```
void( * hsk_isr8_callback::IRDY) (void)
```

Function to be called back when the IRDY interrupt event is triggered.

15.5.2.6 NDOV

```
void( * hsk_isr8_callback::NDOV) (void)
```

Function to be called back when the NDOV interrupt event is triggered.

15.5.2.7 RI

```
void( * hsk_isr8_callback::RI) (void)
```

Function to be called back when the RI interrupt event is triggered.

15.5.2.8 TF2

```
void( * hsk_isr8_callback::TF2) (void)
```

Function to be called back when the TF2 interrupt event is triggered.

15.5.2.9 TI

```
void( * hsk_isr8_callback::TI) (void)
```

Function to be called back when the TI interrupt event is triggered.

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

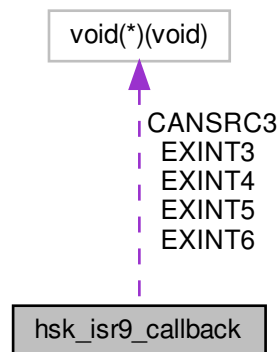
- [hsk_isr/hsk_isr.h](#)

15.6 hsk_isr9_callback Struct Reference

Shared interrupt 9 routine.

```
#include <hsk_isr.h>
```

Collaboration diagram for hsk_isr9_callback:



Data Fields

- void(* [EXINT3](#))(void)
Function to be called back when the EXINT3/T2CC0 interrupt event is triggered.
- void(* [EXINT4](#))(void)
Function to be called back when the EXINT4/T2CC1 interrupt event is triggered.
- void(* [EXINT5](#))(void)
Function to be called back when the EXINT5/T2CC2 interrupt event is triggered.
- void(* [EXINT6](#))(void)
Function to be called back when the EXINT6/T2CC3 interrupt event is triggered.
- void(* [CANSRC3](#))(void)
Function to be called back when the CANSRC3 interrupt event is triggered.

15.6.1 Detailed Description

Shared interrupt 9 routine.

Activate the interrupt by setting EXM = 1.

This interrupt has the following sources:

- EXINT3/T2CC0
- EXINT4/T2CC1
- EXINT5/T2CC2
- EXINT6/T2CC3
- CANSRC2

15.6.2 Field Documentation

15.6.2.1 CANSRC3

```
void( * hsk_isr9_callback::CANSRC3) (void)
```

Function to be called back when the CANSRC3 interrupt event is triggered.

15.6.2.2 EXINT3

```
void( * hsk_isr9_callback::EXINT3) (void)
```

Function to be called back when the EXINT3/T2CC0 interrupt event is triggered.

15.6.2.3 EXINT4

```
void( * hsk_isr9_callback::EXINT4) (void)
```

Function to be called back when the EXINT4/T2CC1 interrupt event is triggered.

15.6.2.4 EXINT5

```
void( * hsk_isr9_callback::EXINT5) (void)
```

Function to be called back when the EXINT5/T2CC2 interrupt event is triggered.

15.6.2.5 EXINT6

```
void( * hsk_isr9_callback::EXINT6) (void)
```

Function to be called back when the EXINT6/T2CC3 interrupt event is triggered.

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

- [hsk_isr/hsk_isr.h](#)

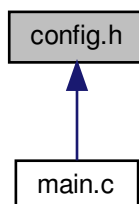
Chapter 16

File Documentation

16.1 config.h File Reference

Configuration for the Infineon XC800 Starter Kit.

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



Macros

- `#define CLK 8000000UL`
The external oscillator clock frequency.
- `#define CAN0_BAUD 1000000`
The CAN0 baud rate in bits/s.
- `#define CAN1_BAUD 1000000`
The CAN1 baud rate in bits/s.
- `#define CAN0_IO CAN0_IO_P10_P11`
The CAN0 IO pin configuration RX P1.0 TX P1.1.
- `#define CAN1_IO CAN1_IO_P14_P13`
The CAN1 IO pin configuration RX P1.4 TX P1.3.

16.1.1 Detailed Description

Configuration for the Infineon XC800 Starter Kit.

Author

kami

16.1.2 Macro Definition Documentation

16.1.2.1 CAN0_BAUD

```
#define CAN0_BAUD 1000000
```

The CAN0 baud rate in bits/s.

16.1.2.2 CAN0_IO

```
#define CAN0_IO CAN0_IO_P10_P11
```

The CAN0 IO pin configuration RX P1.0 TX P1.1.

16.1.2.3 CAN1_BAUD

```
#define CAN1_BAUD 1000000
```

The CAN1 baud rate in bits/s.

16.1.2.4 CAN1_IO

```
#define CAN1_IO CAN1_IO_P14_P13
```

The CAN1 IO pin configuration RX P1.4 TX P1.3.

16.1.2.5 CLK

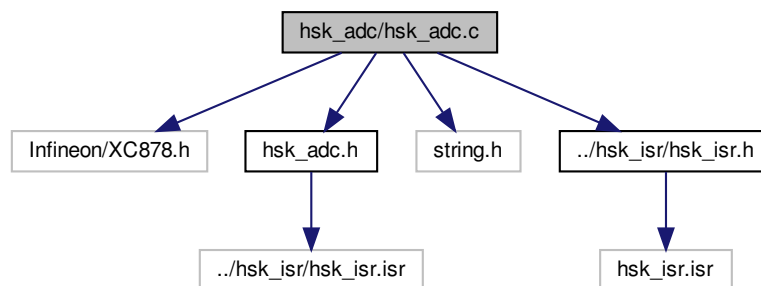
```
#define CLK 8000000UL
```

The external oscillator clock frequency.

16.2 hsk_adc/hsk_adc.c File Reference

HSK Analog Digital Conversion implementation.

```
#include <Infineon/XC878.h>
#include "hsk_adc.h"
#include <string.h>
#include "../hsk_isr/hsk_isr.h"
Include dependency graph for hsk_adc.c:
```



Macros

- `#define ADC_CLK_12MHz 0`
Conversion clock prescaler setting for 12MHz.
- `#define ADC_CLK_8MHz 1`
Conversion clock prescaler setting for 8MHz.
- `#define ADC_CLK_6MHz 2`
Conversion clock prescaler setting for 6MHz.
- `#define ADC_CLK_750kHz 3`
Conversion clock prescaler setting for 750kHz.
- `#define ADC_CHANNELS 8`
Number of availbale ADC channels.
- `#define ADC_QUEUE 4`
Number of queue slots.
- `#define BIT_CHNR 0`
ADC_RESR_xL Channel Number bits.
- `#define CNT_CHNR 3`
CHNR bit count.
- `#define BIT_RESULT 6`
ADC_RESR_xLH Conversion Result bits.
- `#define CNT_RESULT 10`
RESULT bit count.
- `#define BIT_DW 6`
ADC_GLOBCTR Data Width bit.
- `#define BIT CTC 4`
ADC_GLOBCTR Conversion Time Control bits.

- `#define CNT_CTC 2`
CTC bit count.
- `#define BIT_ASEN_SEQUENTIAL 6`
ADC_PRAR Arbitration Slot Sequential Enable bit.
- `#define BIT_ASEN_PARALLEL 7`
ADC_PRAR Arbitration Slot Parallel Enable bit.
- `#define BIT_IEN 4`
RCRx Interrupt Enable bit.
- `#define BIT_WFR 6`
RCRx Wait-for-Read Mode.
- `#define BIT_VFCTR 7`
RCRx Valid Flag Control bit.
- `#define BIT_ENGT 0`
QMR0 Enable Gate bit.
- `#define BIT_ANON 7`
ADC_GLOBCTR Analog Part Switched On bit.
- `#define BIT_IMODE 4`
SYSCON0 Interrupt Structure 2 Mode Select bit.
- `#define BIT_ADC_DIS 0`
PMCON1 ADC Disable Request bit.
- `#define BIT_FILL 0`
QSR0 bits Filling Level.
- `#define CNT_FILL 2`
Filling Level bit count.
- `#define BIT_EMPTY 5`
QSR0 bit Queue Empty.
- `#define BIT_REQCHNR 0`
ADC_QINR0 Request Channel Number bits.
- `#define CNT_REQCHNR 3`
REQCHNR bit count.

Functions

- `void hsk_adc_isr10 (void)`
Write the 10bit conversion result to the targeted memory address.
- `void hsk_adc_isr8 (void)`
Write the 8bit conversion result to the targeted memory address.
- `void hsk_adc_init (ubyte resolution, uword convTime)`
Initialize the AD conversion.
- `void hsk_adc_enable (void)`
Turns on ADC conversion, if previously deactivated.
- `void hsk_adc_disable (void)`
Turns off ADC conversion unit to converse power.
- `void hsk_adc_open10 (const hsk_adc_channel channel, uword *const target)`
Open the given ADC channel in 10 bit mode.
- `void hsk_adc_open8 (const hsk_adc_channel channel, ubyte *const target)`
Open the given ADC channel in 8 bit mode.
- `void hsk_adc_close (const hsk_adc_channel channel)`
Close the given ADC channel.
- `bool hsk_adc_service (void)`

- *A maintenance function that takes care of keeping AD conversions going.*
- bool [hsk_adc_request](#) (const [hsk_adc_channel](#) channel)
Requests an ADC for a specific channel.
- void [hsk_adc_isr_warmup10](#) (void)
Special ISR for warming up 10 bit conversions.
- void [hsk_adc_warmup10](#) (void)
Warm up 10 bit AD conversion.

Variables

- static [hsk_adc_channel](#) nextChannel = 8
Holds the channel of the next conversion that will be requested.
- union {
 uword * [ptr10](#)
 Pointer type used for 10 bit conversions.
 ubyte * [ptr8](#)
 Pointer type used for 8 bit conversions.
} [targets](#) [8]

An array of target addresses to write conversion results into.

16.2.1 Detailed Description

HSK Analog Digital Conversion implementation.

This file implements the functions defined in [hsk_adc.h](#).

To be able to use all 8 channels the ADC is kept in sequential mode.

In order to reduce processing time this library uses the convention that all functions terminate with ADC register page 6. Page 6 contains the ADC queue request and status registers.

Author

kami

16.2.2 Macro Definition Documentation

16.2.2.1 ADC_CHANNELS

```
#define ADC_CHANNELS 8
```

Number of availbale ADC channels.

16.2.2.2 ADC_CLK_12MHz

```
#define ADC_CLK_12MHz 0
```

Conversion clock prescaler setting for 12MHz.

16.2.2.3 ADC_CLK_6MHz

```
#define ADC_CLK_6MHz 2
```

Conversion clock prescaler setting for 6MHz.

16.2.2.4 ADC_CLK_750kHz

```
#define ADC_CLK_750kHz 3
```

Conversion clock prescaler setting for 750kHz.

16.2.2.5 ADC_CLK_8MHz

```
#define ADC_CLK_8MHz 1
```

Conversion clock prescaler setting for 8MHz.

16.2.2.6 ADC_QUEUE

```
#define ADC_QUEUE 4
```

Number of queue slots.

16.2.2.7 BIT_ADC_DIS

```
#define BIT_ADC_DIS 0
```

PMCON1 ADC Disable Request bit.

16.2.2.8 BIT_ANON

```
#define BIT_ANON 7
```

ADC_GLOBCTR Analog Part Switched On bit.

16.2.2.9 BIT_ASEN_PARALLEL

```
#define BIT_ASEN_PARALLEL 7
```

ADC_PRAR Arbitration Slot Parallel Enable bit.

16.2.2.10 BIT_ASEN_SEQUENTIAL

```
#define BIT_ASEN_SEQUENTIAL 6
```

ADC_PRAR Arbitration Slot Sequential Enable bit.

16.2.2.11 BIT_CHNR

```
#define BIT_CHNR 0
```

ADC_RESRxL Channel Number bits.

16.2.2.12 BIT_CTC

```
#define BIT_CTC 4
```

ADC_GLOBCTR Conversion Time Control bits.

16.2.2.13 BIT_DW

```
#define BIT_DW 6
```

ADC_GLOBCTR Data Width bit.

16.2.2.14 BIT_EMPTY

```
#define BIT_EMPTY 5
```

QSR0 bit Queue Empty.

16.2.2.15 BIT_ENGT

```
#define BIT_ENGT 0
```

QMR0 Enable Gate bit.

16.2.2.16 BIT_FILL

```
#define BIT_FILL 0
```

QSR0 bits Filling Level.

16.2.2.17 BIT_IEN

```
#define BIT_IEN 4
```

RCRx Interrupt Enable bit.

16.2.2.18 BIT_IMODE

```
#define BIT_IMODE 4
```

SYSCON0 Interrupt Structure 2 Mode Select bit.

16.2.2.19 BIT_REQCHNR

```
#define BIT_REQCHNR 0
```

ADC_QINR0 Request Channel Number bits.

16.2.2.20 BIT_RESULT

```
#define BIT_RESULT 6
```

ADC_RESRxLH Conversion Result bits.

16.2.2.21 BIT_VFCTR

```
#define BIT_VFCTR 7
```

RCRx Valid Flag Control bit.

16.2.2.22 BIT_WFR

```
#define BIT_WFR 6
```

RCRx Wait-for-Read Mode.

16.2.2.23 CNT_CHNR

```
#define CNT_CHNR 3
```

CHNR bit count.

16.2.2.24 CNT CTC

```
#define CNT_CTC 2
```

CTC bit count.

16.2.2.25 CNT_FILL

```
#define CNT_FILL 2
```

Filling Level bit count.

16.2.2.26 CNT_REQCHNR

```
#define CNT_REQCHNR 3
```

REQCHNR bit count.

16.2.2.27 CNT_RESULT

```
#define CNT_RESULT 10
```

RESULT bit count.

16.2.3 Function Documentation

16.2.3.1 hsk_adc_close()

```
void hsk_adc_close (
    const hsk_adc_channel channel )
```

Close the given ADC channel.

Stopp ADC if no more channels were left.

Parameters

<i>channel</i>	The channel id
----------------	----------------

16.2.3.2 hsk_adc_disable()

```
void hsk_adc_disable (
    void )
```

Turns off ADC conversion unit to converse power.

16.2.3.3 hsk_adc_enable()

```
void hsk_adc_enable (
    void )
```

Turns on ADC conversion, if previously deactivated.

16.2.3.4 hsk_adc_init()

```
void hsk_adc_init (
    ubyte resolution,
    uword convTime )
```

Initialize the AD conversion.

The shortest possible conversion time is 1.25µs, the longest is 714.75µs. The given value will be rounded down.

Note if [hsk_adc_service\(\)](#) is not called in intervals shorter than convTime, there will be a waiting period between conversions. This prevents locking up of the controller with erratic interrupts.

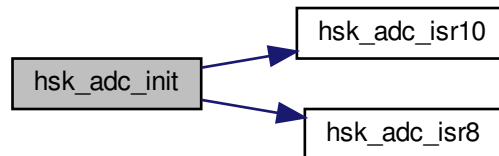
There is a 4 entry queue, for starting conversions, so it suffices to average the interval below convTime.

All already open channels will be closed upon calling this function.

Parameters

<i>resolution</i>	The conversion resolution, any of ADC_RESOLUTION_*
<i>convTime</i>	The desired conversion time in μ s

Here is the call graph for this function:



16.2.3.5 hsk_adc_isr10()

```
void hsk_adc_isr10 (
    void ) [private]
```

Write the 10bit conversion result to the targeted memory address.

16.2.3.6 hsk_adc_isr8()

```
void hsk_adc_isr8 (
    void ) [private]
```

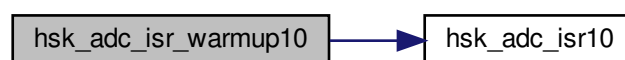
Write the 8bit conversion result to the targeted memory address.

16.2.3.7 hsk_adc_isr_warmup10()

```
void hsk_adc_isr_warmup10 (
    void ) [private]
```

Special ISR for warming up 10 bit conversions.

This is used as the ISR by [hsk_adc_warmup\(\)](#) after the warmup countdowns have been initialized. After all warmup countdowns have returned to zero The original ISR will be put back in control. Here is the call graph for this function:



16.2.3.8 hsk_adc_open10()

```
void hsk_adc_open10 (
    const hsk_adc_channel channel,
    uword *const target )
```

Open the given ADC channel in 10 bit mode.

Parameters

<i>channel</i>	The channel id
<i>target</i>	A pointer where to store conversion results

16.2.3.9 hsk_adc_open8()

```
void hsk_adc_open8 (
    const hsk_adc_channel channel,
    ubyte *const target )
```

Open the given ADC channel in 8 bit mode.

Parameters

<i>channel</i>	The channel id
<i>target</i>	A pointer where to store conversion results

16.2.3.10 hsk_adc_request()

```
bool hsk_adc_request (
    const hsk_adc_channel channel )
```

Requests an ADC for a specific channel.

This function is an alternative to [hsk_adc_service\(\)](#). Make requests in time before the updated value is required.

This function uses the same queue as [hsk_adc_service\(\)](#), if the queue is full it fails silently.

Parameters

<i>channel</i>	The channel id
----------------	----------------

Return values

0	The queue is full
1	A conversion request has been added to the queue

16.2.3.11 hsk_adc_service()

```
bool hsk_adc_service (
    void )
```

A maintenance function that takes care of keeping AD conversions going.

This has to be called repeatedly.

There is a queue of up to 4 conversion jobs. One call of this function only adds one job to the queue.

Return values

0	No conversion request had been queued, either the queue is full or no channels have been configured
1	A conversion request has been added to the queue

Here is the call graph for this function:



16.2.3.12 hsk_adc_warmup10()

```
void hsk_adc_warmup10 (
    void )
```

Warm up 10 bit AD conversion.

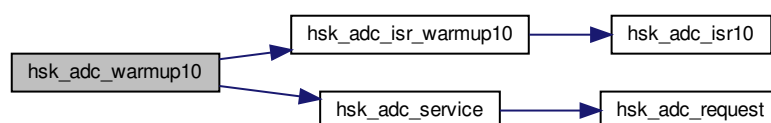
I.e. make sure all conversion targets have been initialized with a conversion result. This is a blocking function only intended for single use during the boot procedure.

This function will not terminate unless interrupts are enabled.

Note

This function only works in 10 bit mode, because in 8 bit mode it is impossible to initialize targets with an invalid value.

Here is the call graph for this function:



16.2.4 Variable Documentation

16.2.4.1 nextChannel

```
hsk_adc_channel nextChannel = 8 [static]
```

Holds the channel of the next conversion that will be requested.

16.2.4.2 ptr10

```
uword* ptr10
```

Pointer type used for 10 bit conversions.

16.2.4.3 ptr8

```
ubyte* ptr8
```

Pointer type used for 8 bit conversions.

16.2.4.4 targets

```
targets [static]
```

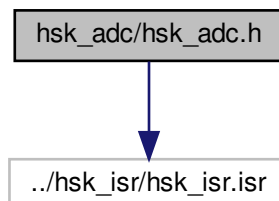
An array of target addresses to write conversion results into.

16.3 hsk_adc/hsk_adc.h File Reference

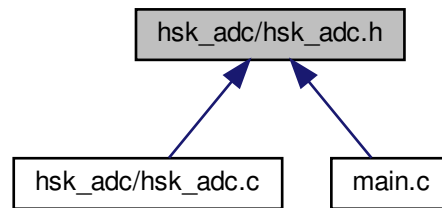
HSK Analog Digital Conversion headers.

```
#include "../hsk_isr/hsk_isr.isr"
```

Include dependency graph for hsk_adc.h:



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



Macros

- `#define ADC_RESOLUTION_10 0`
10 bit ADC resolution.
- `#define ADC_RESOLUTION_8 1`
8 bit ADC resolution.
- `#define hsk_adc_open hsk_adc_open10`
Backwards compatibility hack.
- `#define hsk_adc_warmup hsk_adc_warmup10`
Backwards compatibility hack.

Typedefs

- `typedef ubyte hsk_adc_channel`
Typedef for ADC channel ids.

Functions

- `void hsk_adc_init (ubyte resolution, uword convTime)`
Initialize the AD conversion.
- `void hsk_adc_enable (void)`
Turns on ADC conversion, if previously deactivated.
- `void hsk_adc_disable (void)`
Turns off ADC conversion unit to converse power.
- `void hsk_adc_open10 (const hsk_adc_channel channel, uword *const target)`
Open the given ADC channel in 10 bit mode.
- `void hsk_adc_open8 (const hsk_adc_channel channel, ubyte *const target)`
Open the given ADC channel in 8 bit mode.
- `void hsk_adc_close (const hsk_adc_channel channel)`
Close the given ADC channel.
- `bool hsk_adc_service (void)`
A maintenance function that takes care of keeping AD conversions going.
- `bool hsk_adc_request (const hsk_adc_channel channel)`
Requests an ADC for a specific channel.
- `void hsk_adc_warmup10 (void)`
Warm up 10 bit AD conversion.

16.3.1 Detailed Description

HSK Analog Digital Conversion headers.

This library provides access to all 8 ADC channels. Each channel can be provided with a pointer. Every completed conversion is written to the address provided by the pointer. The target memory can be protected for read access by masking the interrupts with EADC.

The conversion time can be freely configured in a wide range. Even short conversion times like 5µs yield good precision.

In order to keep the conversion going a service function [hsk_adc_service\(\)](#) has to be called on a regular basis. This prevents locking up of the CPU due to an overload of interrupts, the ADC module can provide a new conversion result every 30 clock cycles.

Making the [hsk_adc_service\(\)](#) call only as often as needed reduces the drain on the analogue input and reduces flickering.

Alternatively [hsk_adc_request\(\)](#) can be used to request single just in time conversions.

Author

kami

16.3.2 Macro Definition Documentation

16.3.2.1 ADC_RESOLUTION_10

```
#define ADC_RESOLUTION_10 0
```

10 bit ADC resolution.

16.3.2.2 ADC_RESOLUTION_8

```
#define ADC_RESOLUTION_8 1
```

8 bit ADC resolution.

16.3.2.3 hsk_adc_open

```
#define hsk_adc_open hsk\_adc\_open10
```

Backwards compatibility hack.

Deprecated Use [hsk_adc_open10\(\)](#) or [hsk_adc_open8\(\)](#) as appropriate

16.3.2.4 hsk_adc_warmup

```
#define hsk_adc_warmup hsk_adc_warmup10
```

Backwards compatibility hack.

Deprecated Use `hsk_adc_warmup10()`

16.3.3 Typedef Documentation

16.3.3.1 hsk_adc_channel

```
typedef ubyte hsk_adc_channel
```

Typedef for ADC channel ids.

16.3.4 Function Documentation

16.3.4.1 hsk_adc_close()

```
void hsk_adc_close (
    const hsk_adc_channel channel )
```

Close the given ADC channel.

Stopp ADC if no more channels were left.

Parameters

<i>channel</i>	The channel id
----------------	----------------

16.3.4.2 hsk_adc_disable()

```
void hsk_adc_disable (
    void )
```

Turns off ADC conversion unit to converse power.

16.3.4.3 hsk_adc_enable()

```
void hsk_adc_enable (
    void )
```

Turns on ADC conversion, if previously deactivated.

16.3.4.4 hsk_adc_init()

```
void hsk_adc_init (
    ubyte resolution,
    uword convTime )
```

Initialize the AD conversion.

The shortest possible conversion time is 1.25µs, the longest is 714.75µs. The given value will be rounded down.

Note if [hsk_adc_service\(\)](#) is not called in intervals shorter than convTime, there will be a waiting period between conversions. This prevents locking up of the controller with erratic interrupts.

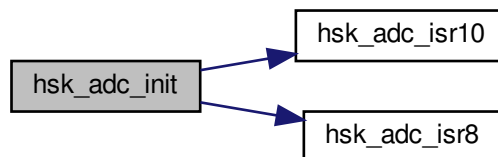
There is a 4 entry queue, for starting conversions, so it suffices to average the interval below convTime.

All already open channels will be closed upon calling this function.

Parameters

<i>resolution</i>	The conversion resolution, any of ADC_RESOLUTION_*
<i>convTime</i>	The desired conversion time in µs

Here is the call graph for this function:



16.3.4.5 hsk_adc_open10()

```
void hsk_adc_open10 (
    const hsk_adc_channel channel,
    uword *const target )
```

Open the given ADC channel in 10 bit mode.

Parameters

<i>channel</i>	The channel id
<i>target</i>	A pointer where to store conversion results

16.3.4.6 hsk_adc_open8()

```
void hsk_adc_open8 (
    const hsk_adc_channel channel,
    ubyte *const target )
```

Open the given ADC channel in 8 bit mode.

Parameters

<i>channel</i>	The channel id
<i>target</i>	A pointer where to store conversion results

16.3.4.7 hsk_adc_request()

```
bool hsk_adc_request (
    const hsk_adc_channel channel )
```

Requests an ADC for a specific channel.

This function is an alternative to [hsk_adc_service\(\)](#). Make requests in time before the updated value is required.

This function uses the same queue as [hsk_adc_service\(\)](#), if the queue is full it fails silently.

Parameters

<i>channel</i>	The channel id
----------------	----------------

Return values

0	The queue is full
1	A conversion request has been added to the queue

16.3.4.8 hsk_adc_service()

```
bool hsk_adc_service (
    void )
```

A maintenance function that takes care of keeping AD conversions going.

This has to be called repeatedly.

There is a queue of up to 4 conversion jobs. One call of this function only adds one job to the queue.

Return values

0	No conversion request had been queued, either the queue is full or no channels have been configured
1	A conversion request has been added to the queue

Here is the call graph for this function:



16.3.4.9 hsk_adc_warmup10()

```
void hsk_adc_warmup10 (  
    void )
```

Warm up 10 bit AD conversion.

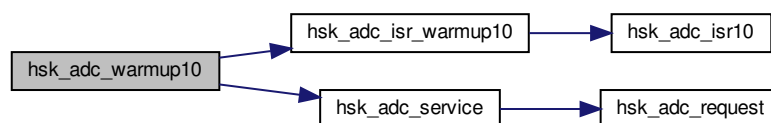
I.e. make sure all conversion targets have been initialized with a conversion result. This is a blocking function only intended for single use during the boot procedure.

This function will not terminate unless interrupts are enabled.

Note

This function only works in 10 bit mode, because in 8 bit mode it is impossible to initialize targets with an invalid value.

Here is the call graph for this function:



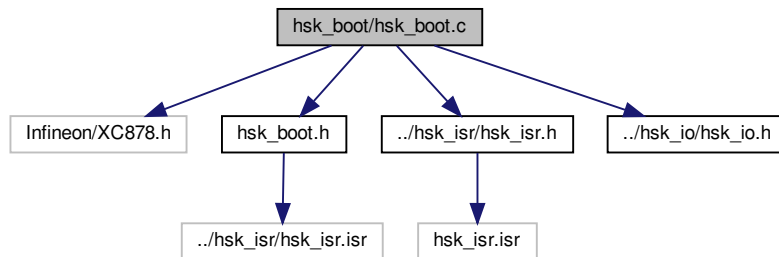
16.4 hsk_boot/hsk_boot.c File Reference

HSK Boot implementation.

```
#include <Infineon/XC878.h>  
#include "hsk_boot.h"  
#include "../hsk_isr/hsk_isr.h"
```

```
#include "../hsk_io/hsk_io.h"
```

Include dependency graph for hsk_boot.c:



Macros

- `#define BIT_MXB 0`
MEX3 XRAM Bank Number bits.
- `#define CNT_MXB 3`
MEX3 XRAM Bank Number bit count.
- `#define BIT_MXB19 4`
MEX3 XRAM Bank Number highest bit.
- `#define XRAM_BANK 0xF`
The selected XRAM bank number.
- `#define BIT_MXM 3`
MEX3 XRAM Bank Selector bit.
- `#define XRAM_SELECTOR 1`
Set BIT_MXM to access the data memroy bank with MOVX instructions.
- `#define PDATA_PAGE 0xF0`
The page to locate pdata at.
- `#define BIT_EXTOSCR 0`
OSC_CON bit.
- `#define BIT_EORDRES 1`
OSC_CON bit.
- `#define BIT_OSCSS 2`
OSC_CON bit.
- `#define BIT_XPD 3`
OSC_CON bit.
- `#define BIT_PLLPD 5`
OSC_CON bit.
- `#define BIT_PLLBYP 6`
OSC_CON bit.
- `#define BIT_PLLRDRES 7`
OSC_CON bit.
- `#define BIT_PLL_LOCK 0`
PLL_CON bit.
- `#define BIT_PLLR 1`
PLL_CON bit.

- #define `BIT_PDIV` 0
PLL_CON1 bit.
- #define `CNT_PDIV` 5
PDIV bit count.
- #define `BIT_NDIVL` 2
PLL_CON low PLL NF-Divider bits.
- #define `CNT_NDIVL` 6
NDIVL bit count.
- #define `BIT_NDIVH` 5
PLL_CON1 high PLL NF-Divider bits.
- #define `CNT_NDIVH` 3
NDIVH bit count.
- #define `BIT_NMIPLL` 1
NMICON PLL Loss of Clock NMI Enable bit.

Functions

- void `hsk_boot_io` (void)
Initialises all IO ports as input ports without pull.
- void `hsk_boot_mem` (void)
Sets up xdata and pdata memory access.
- ubyte `_sdcc_external_startup` (void)
Turns off pullup/-down for all ports prior to global/static initialisation.
- void `hsk_boot_isr_nmipll` (void)
Loss of clock recovery ISR.
- void `hsk_boot_extClock` (const ulong clk)
Switches to an external oscillator.

Variables

- struct {
 ubyte `pdiv`
 The PDIV value for the configured clock speed.
 uword `ndiv`
 The NDIV value for the configured clock speed.
} `boot`

Boot parameter storage for the loss of clock ISR callback.

16.4.1 Detailed Description

HSK Boot implementation.

The High Speed Karlsruhe XC878 boot up code implementation.

This obsoletes 3rd party provided assembler boot code.

Author

kami

16.4.2 Macro Definition Documentation

16.4.2.1 BIT_EORDRES

```
#define BIT_EORDRES 1
```

OSC_CON bit.

External Oscillator Watchdog Reset, used when switching to an external clock.

16.4.2.2 BIT_EXTOSCR

```
#define BIT_EXTOSCR 0
```

OSC_CON bit.

External Oscillator Run Status Bit, used to determine whether the external oscillator is available.

16.4.2.3 BIT_MXB

```
#define BIT_MXB 0
```

MEX3 XRAM Bank Number bits.

Used to select the memory bank where the XRAM is located. This 4 bit field is divided, the highest bit goes into the BIT_MXB19 bit.

16.4.2.4 BIT_MXB19

```
#define BIT_MXB19 4
```

MEX3 XRAM Bank Number highest bit.

The final MXB bit.

16.4.2.5 BIT_MXM

```
#define BIT_MXM 3
```

MEX3 XRAM Bank Selector bit.

16.4.2.6 BIT_NDIVH

```
#define BIT_NDIVH 5
```

PLL_CON1 high PLL NF-Divider bits.

16.4.2.7 BIT_NDIVL

```
#define BIT_NDIVL 2
```

PLL_CON low PLL NF-Divider bits.

16.4.2.8 BIT_NMIPLL

```
#define BIT_NMIPLL 1
```

NMICON PLL Loss of Clock NMI Enable bit.

16.4.2.9 BIT_OSCSS

```
#define BIT_OSCSS 2
```

OSC_CON bit.

Oscillator Source Select, used to turn the external oscillator on(1)/off(0).

16.4.2.10 BIT_PDIV

```
#define BIT_PDIV 0
```

PLL_CON1 bit.

Something to do with the CPU clock.

16.4.2.11 BIT_PLL_LOCK

```
#define BIT_PLL_LOCK 0
```

PLL_CON bit.

PLL Lock Status Flag, used when switching to an external clock.

16.4.2.12 BIT_PLLBYP

```
#define BIT_PLLBYP 6
```

OSC_CON bit.

PLL Output Bypass Control, used when switching to an external clock.

16.4.2.13 BIT_PLLPD

```
#define BIT_PLLPD 5
```

OSC_CON bit.

PLL Power Down Control, used when switching to an external clock.

16.4.2.14 BIT_PLLR

```
#define BIT_PLLR 1
```

PLL_CON bit.

PLL Run Status Flag, used when switching to an external clock.

16.4.2.15 BIT_PLLRDRES

```
#define BIT_PLLRDRES 7
```

OSC_CON bit.

PLL Watchdog Reset, used when switching to an external clock.

16.4.2.16 BIT_XPD

```
#define BIT_XPD 3
```

OSC_CON bit.

XTAL Power Down Control, used when switching to an external clock.

16.4.2.17 CNT_MXB

```
#define CNT_MXB 3
```

MEX3 XRAM Bank Number bit count.

16.4.2.18 CNT_NDIVH

```
#define CNT_NDIVH 3
```

NDIVH bit count.

16.4.2.19 CNT_NDIVL

```
#define CNT_NDIVL 6
```

NDIVL bit count.

16.4.2.20 CNT_PDIV

```
#define CNT_PDIV 5
```

PDIV bit count.

16.4.2.21 PDATA_PAGE

```
#define PDATA_PAGE 0xF0
```

The page to locate pdata at.

Use the first XRAM page, because that is where the compilers expect it.

16.4.2.22 XRAM_BANK

```
#define XRAM_BANK 0xF
```

The selected XRAM bank number.

16.4.2.23 XRAM_SELECTOR

```
#define XRAM_SELECTOR 1
```

Set BIT_MXM to access the data memroy bank with MOVX instructions.

Otherwise the current bank (whichever that is) would be addressed. MOVX is used to access external memory. The data memory bank is selected with the MXB bits.

16.4.3 Function Documentation

16.4.3.1 _sdcc_external_startup()

```
ubyte _sdcc_external_startup (  
    void ) [private]
```

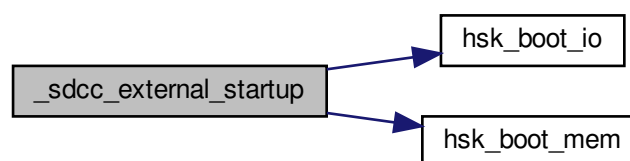
Turns off pullup/-down for all ports prior to global/static initialisation.

This function is automatically linked by SDCC and called from startup.a51 by Keil C51.

Returns

Always returns 0, which indicates that SDCC should initialise globals and statics

Here is the call graph for this function:



16.4.3.2 hsk_boot_extClock()

```
void hsk_boot_extClock (
    const unsigned long clk )
```

Switches to an external oscillator.

This function requires xdata access.

The implemented process is named: "Select the External Oscillator as PLL input source"

The following is described in more detail in chapter 7.3 of the XC878 User Manual.

The XC878 can either use an internal 4MHz oscillator (default) or an external oscillator from 2 to 20MHz, normally referred to as FOSC. A phase-locked loop (PLL) converts it to a faster internal speed FSYS, 144MHz by default.

This implementation is currently limited to oscillators from 2MHz to 20MHz in 1MHz intervals.

The oscillator frequency is vital for external communication (e.g. CAN) and timer/counter speeds.

This implementation switches to an external clock ensuring that the PLL generates a 144MHz FSYS clock. The CLKREL divisor set to 6 generates the fast clock (FCLK) that runs at 48MHz. The remaining clocks, i.e. peripheral (PCLK), CPU (SCLK, CCLK), have a fixed divisor by 2, so they run at 24MHz.

After setting up the PLL, this function will register an ISR, that will attempt to reactivate the external oscillator in a PLL loss-of-clock event.

Parameters

<i>clk</i>	The frequency of the external oscillator in Hz.
------------	---

WARNING - Here be dragons ...

Before messing with this stuff you should be aware that this is tricky business. Mistakes can result in hardware damage. Or at least all your timers and external interfaces will act weird.

Basically this bypasses/turns off the PLL, sets up the external oscillator and then reconfigures the PLL and brings it back into play.

Many of the OSC_CON, which is on page 1, bits are write protected. The MAIN_vUnlockProtectReg() turns the protection off for 32 cycles. So it has to be turned off each time protected bits are accessed. Here is the call graph for this function:



16.4.3.3 hsk_boot_io()

```
void hsk_boot_io (
    void ) [private]
```

Initialises all IO ports as input ports without pull.

16.4.3.4 hsk_boot_isr_nmipll()

```
void hsk_boot_isr_nmipll (
    void ) [private]
```

Loss of clock recovery ISR.

This takes very long.

16.4.3.5 hsk_boot_mem()

```
void hsk_boot_mem (
    void ) [private]
```

Sets up xdata and pdata memory access.

Refer to the Processor Architecture and Memory Organization chapters of the XC878 User Manual.

16.4.4 Variable Documentation

16.4.4.1 boot

```
boot [static]
```

Boot parameter storage for the loss of clock ISR callback.

16.4.4.2 ndiv

```
uword ndiv
```

The NDIV value for the configured clock speed.

See table 7-5 in the data sheet for desired NDIV values. See the NDIV description for value encoding.

16.4.4.3 pdiv

```
ubyte pdiv
```

The PDIV value for the configured clock speed.

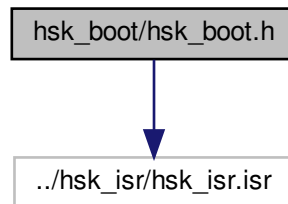
See table 7-5 in the data sheet for desired PDIV values. See the PDIV description for value encoding.

16.5 hsk_boot/hsk_boot.h File Reference

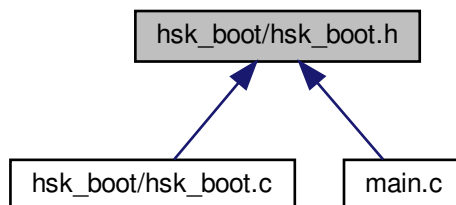
HSK Boot headers.

```
#include "../hsk_isr/hsk_isr.isr"
```

Include dependency graph for hsk_boot.h:



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



Functions

- void [hsk_boot_extClock](#) (const ulong clk)
Switches to an external oscillator.

16.5.1 Detailed Description

HSK Boot headers.

This file contains the prototypes to put the μ C into working condition.

Currently implemented:

- [hsk_boot_extClock\(\)](#) Activates external clock input and sets up the PLL, this is important when communicating with other devices, the internal clock is not sufficiently precise

Linking this library also automatically causes the following boot actions:

- Deactivate all internal pullup devices
- Activate XDATA access
- Set the PDATA page to the first XDATA block

Author

kami

16.5.2 Function Documentation

16.5.2.1 hsk_boot_extClock()

```
void hsk_boot_extClock (
    const ulong clk )
```

Switches to an external oscillator.

This function requires xdata access.

The implemented process is named: "Select the External Oscillator as PLL input source"

The following is described in more detail in chapter 7.3 of the XC878 User Manual.

The XC878 can either use an internal 4MHz oscillator (default) or an external oscillator from 2 to 20MHz, normally referred to as FOSC. A phase-locked loop (PLL) converts it to a faster internal speed FSYS, 144MHz by default.

This implementation is currently limited to oscillators from 2MHz to 20MHz in 1MHz intervals.

The oscillator frequency is vital for external communication (e.g. CAN) and timer/counter speeds.

This implementation switches to an external clock ensuring that the PLL generates a 144MHz FSYS clock. The CLKREL divisor set to 6 generates the fast clock (FCLK) that runs at 48MHz. The remaining clocks, i.e. peripheral (PCLK), CPU (SCLK, CCLK), have a fixed divisor by 2, so they run at 24MHz.

After setting up the PLL, this function will register an ISR, that will attempt to reactivate the external oscillator in a PLL loss-of-clock event.

Parameters

<i>clk</i>	The frequency of the external oscillator in Hz.
------------	---

WARNING - Here be dragons ...

Before messing with this stuff you should be aware that this is tricky business. Mistakes can result in hardware damage. Or at least all your timers and external interfaces will act weird.

Basically this bypasses/turns off the PLL, sets up the external oscillator and then reconfigures the PLL and brings it back into play.

Many of the OSC_CON, which is on page 1, bits are write protected. The MAIN_vUnlockProtecReg() turns the protection off for 32 cycles. So it has to be turned off each time protected bits are accessed. Here is the call graph for this function:

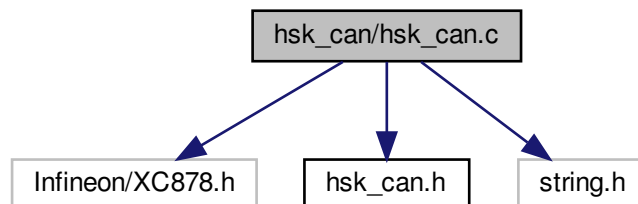


16.6 hsk_can/hsk_can.c File Reference

HSK Controller Area Network implementation.

```
#include <Infineon/XC878.h>
#include "hsk_can.h"
#include <string.h>
```

Include dependency graph for hsk_can.c:



Macros

- `#define BIT_RWEN 0`
CAN_ADCON Read/Write Enable bit.
- `#define BIT_BSY 1`
CAN_ADCON Data Transmission Busy bit.
- `#define BIT_AUAD 2`
CAN_ADCON Auto Increment/Decrement the Address bits.
- `#define AUAD_OFF (0 << BIT_AUAD)`
AUAD auto increment off setting.
- `#define AUAD_INC1 (1 << BIT_AUAD)`
AUAD auto increment setting.

- #define `AUAD_DEC1` (2 << `BIT_AUAD`)
AUAD auto decrement setting.
- #define `AUAD_INC8` (3 << `BIT_AUAD`)
AUAD auto increment setting.
- #define `BIT_DATA` 4
CAN_ADCON CAN Data Valid bits.
- #define `CNT_DATA` 4
DATA bit count.
- #define `CAN_AD_WRITE`(msk) `CAN_ADCON = (1 << BIT_RWEN) | ((msk) << BIT_DATA)`
Sets up the CAN_AD bus for writing.
- #define `CAN_AD_READ`() `CAN_ADCON = 0`
Sets up the CAN_AD bus for reading.
- #define `CAN_AD_READY`() while (`CAN_ADCON & (1 << BIT_BSY)`)
Make sure the last read/write has completed.
- #define `BIT_FCCFG` 4
CMCON MultiCAN Clock Configuration bit.
- #define `OFF_LISTm` 0
The Id() of the List Register (LISTm) m offset factor.
- #define `OFF_MSIDk` 0
The Id() of the Message Index Register k offset factor.
- #define `OFF_MSPNDk` 0
The Id() of the Message Pending Register k offset factor.
- #define `OFF_NODEx` 6
The Id() of the Node Register x offset factor.
- #define `OFF_MOn` 3
The Id() of the Message Object n offset factor.
- #define `NCRx` 0x0080
Node x Control Register base address.
- #define `NSRx` 0x0081
Node x Status Register base address.
- #define `NIPRx` 0x0082
Node x Interrupt Pointer Register base address.
- #define `NPCRx` 0x0083
Node x Port Control Register base address.
- #define `NBTRx` 0x0084
Node x Bit Timing Register base address.
- #define `NECNTx` 0x0085
Node x Error Counter Register base address.
- #define `NFCRx` 0x0086
Node x Frame Counter Register base address.
- #define `BIT_INIT` 0
CAN NCRx Node Initialization bit.
- #define `BIT_TRIE` 1
CAN NCRx Transfer Interrupt Enable bit.
- #define `BIT_LECIE` 2
CAN NCRx LEC Indicated Error Interrupt Enable bit.
- #define `BIT_ALIE` 3
CAN NCRx Alert Interrupt Enable bit.
- #define `BIT_CANDIS` 4
CAN NCRx CAN Disable.
- #define `BIT_CCE` 6

- *CAN NCRx Configuration Change Enable bit.*
- #define [BIT_CALM](#) 7
- *CAN NCRx CAN Analyze Mode bit.*
- #define [BIT_BRP](#) 0
- *NBTRx Baud Rate Prescaler bits.*
- #define [BIT_SJW](#) 6
- *NBTRx (Re) Synchronization Jump Width bits.*
- #define [BIT_TSEG1](#) 8
- *NBTRx Time Segment Before Sample Point bits.*
- #define [BIT_TSEG2](#) 12
- *NBTRx Time Segment After Sample Point bits.*
- #define [BIT_DIV8](#) 15
- *NBTRx Divide Prescaler Clock by 8 bit.*
- #define [BIT_RXSEL](#) 0
- *NPCRx Receive Select bit.*
- #define [CNT_RXSEL](#) 3
- *RXSEL bit count.*
- #define [PANCTR](#) 0x0071
- *The Panel Control Register.*
- #define [PANCMD](#) CAN_DATA0
- *PANCTR Command Register.*
- #define [PANSTATUS](#) CAN_DATA1
- *PANCTR Status Register.*
- #define [BIT_BUSY](#) 0
- *PANCTR PANSTATUS Panel Busy Flag bit.*
- #define [BIT_RBUSY](#) 1
- *PANCTR PANSTATUS Result Busy Flag bit.*
- #define [PANAR1](#) CAN_DATA2
- *PANCTR Argument 1 Register.*
- #define [PANAR2](#) CAN_DATA3
- *PANCTR Argument 2 Register.*
- #define [BIT_ERR](#) 7
- *PANCTR PANAR2 Error bit.*
- #define [PANCTR_READY](#)()
- *Wait for list operations to complete.*
- #define [PAN_CMD_NOP](#) 0x00
- *List panel No Operation command.*
- #define [PAN_CMD_INIT](#) 0x01
- *List panel Initialize Lists command.*
- #define [PAN_CMD_MOVE](#) 0x02
- *List panel Static Allocate command.*
- #define [PAN_CMD_ALLOC](#) 0x03
- *List panel Dynamic Allocate command.*
- #define [PAN_CMD_MOVEBEFORE](#) 0x04
- *List panel Static Insert Before command.*
- #define [PAN_CMD_ALLOCBEFORE](#) 0x05
- *List panel Dynamic Insert Before command.*
- #define [PAN_CMD_MOVEBEHIND](#) 0x06
- *List panel Static Insert Behind command.*
- #define [PAN_CMD_ALLOCBEHIND](#) 0x07
- *List panel Dynamic Insert Behind command.*

- #define `HSK_CAN_MSG_MAX` 32
The maximum number of message objects.
- #define `LIST_UNALLOC` 0
This list holds unallocated message objects.
- #define `LIST_NODEx` 1
These lists hold message objects connected to a CAN node.
- #define `LIST_PENDING` 3
This list holds message objects pending assignment to a can node.
- #define `BIT_CAN_DIS` 5
PMCON1 CAN Disable Request bit.
- #define `MOFCRn` 0x0400
Message Object n Function Control Register base address.
- #define `MOFGPRn` 0x0401
Message Object n FIFO/Gateway Pointer Register base address.
- #define `MOAMRn` 0x0403
Message Object n Acceptance Mask Register base address.
- #define `MODATALn` 0x0404
Message Object n Data Register Low base address.
- #define `MODATAHn` 0x0405
Message Object n Data Register High base address.
- #define `MOARn` 0x0406
Message Object n Arbitration Register base address.
- #define `MOCTRn` 0x0407
Message Object n Control Register base address.
- #define `MOSTATn` `MOCTRn`
Message Object n Status Register base address.
- #define `RESET_DATA` `CAN_DATA01`
The register to write Control Register resets into.
- #define `SET_DATA` `CAN_DATA23`
The register to write Control Register settings into.
- #define `RESET` 0x3
Bit mask for writing resets.
- #define `SET` 0xC
Bit mask for writing settings.
- #define `BIT_RXPND` 0
MOCTRn/MOSTATn Receive Pending bit.
- #define `BIT_TXPND` 1
MOCTRn/MOSTATn Transmit Pending bit.
- #define `BIT_RXUPD` 2
MOCTRn/MOSTATn Receive Updating bit.
- #define `BIT_NEWDAT` 3
MOCTRn/MOSTATn New Data bit.
- #define `BIT_MSGVAL` 5
MOCTRn/MOSTATn Message Valid bit.
- #define `BIT_RXEN` 7
MOCTRn/MOSTATn Receive Signal Enable bit.
- #define `BIT_TXRQ` 8
MOCTRn/MOSTATn Transmit Signal Request bit.
- #define `BIT_TXEN0` 9
MOCTRn/MOSTATn Transmit Signal Enable bit.
- #define `BIT_TXEN1` 10

- *MOCTRn/MOSTATn Transmit Signal Enable Select bit.*
- #define `BIT_DIR` 11
 - *MOCTRn Direction bit.*
- #define `BIT_AM` 0
 - *MOAMRn Acceptance Mask for Message Identifier bits.*
- #define `CNT_AM` 29
 - *AM bit count.*
- #define `BIT_MIDE` 29
 - *MOAMRn Acceptance Mask Bit for Message IDE Bit.*
- #define `BIT_DLC` 0
 - *MOFCRn Data Length Code bits in byte 3.*
- #define `CNT_DLC` 4
 - *DLC bit count.*
- #define `BIT_MMC` 0
 - *MOFCRn Message Mode Control bits in byte 0.*
- #define `CNT_MMC` 4
 - *MMC bit count.*
- #define `MMC_DEFAULT` 0
 - *Regular message mode.*
- #define `MMC_RXBASEFIFO` 1
 - *Message is the base of an RX FIFO.*
- #define `MMC_TXBASEFIFO` 2
 - *Message is the base of a TX FIFO.*
- #define `MMC_TXSLAVEFIFO` 3
 - *Message is a TX FIFO slave.*
- #define `MMC_GATEWAYSRC` 4
 - *Message is a source object for a gateway.*
- #define `BIT_IDEXT` 0
 - *MOARn Extended CAN Identifier of Message Object n bits.*
- #define `CNT_IDEXT` 29
 - *ID bit count.*
- #define `BIT_IDSTD` 18
 - *MOARn Standard CAN Identifier of Message Object n bits.*
- #define `CNT_IDSTD` 11
 - *ID bit count.*
- #define `BIT_IDE` 29
 - *MOARn Identifier Extension Bit of Message Object n.*
- #define `BIT_PRI` 30
 - *MOARn Priority Class bits.*
- #define `CNT_PRI` 2
 - *PRI bit count.*
- #define `PRI_LIST` 1
 - *List order based transmit priority.*
- #define `PRI_ID` 2
 - *CAN ID based transmit priority.*
- #define `BIT_LIST` 4
 - *MOSTATn List Allocation bits in byte 1.*
- #define `CNT_LIST` 4
 - *LIST bit count.*
- #define `MOSTATn_PNEXT` CAN_DATA3
 - *MOSTATn Pointer to Next Message Object byte.*

- `#define MOFGPRn_BOT CAN_DATA0`
MOFGPRn bottom pointer byte.
- `#define MOFGPRn_TOP CAN_DATA1`
MOFGPRn top pointer byte.
- `#define MOFGPRn_CUR CAN_DATA2`
MOFGPRn current pointer byte.
- `#define MOFGPRn_SEL CAN_DATA3`
MOFGPRn select pointer byte.

Functions

- `void hsk_can_init (const ubyte pins, const ulong baud)`
Setup CAN communication with the desired baud rate.
- `void hsk_can_enable (const hsk_can_node node)`
Go live on the CAN bus.
- `void hsk_can_disable (const hsk_can_node node)`
Disable a CAN node.
- `ubyte hsk_can_status (const hsk_can_node node, const ubyte field)`
Returns a status field of a CAN node.
- `hsk_can_msg hsk_can_msg_create (const ulong id, const bool extended, const ubyte dlc)`
Creates a new CAN message.
- `ubyte hsk_can_msg_move (const hsk_can_msg msg, const ubyte list)`
Move the selected message and its slaves to a different list.
- `ubyte hsk_can_msg_connect (const hsk_can_msg msg, const hsk_can_node node)`
Connect a message object to a CAN node.
- `ubyte hsk_can_msg_disconnect (const hsk_can_msg msg)`
Disconnect a CAN message object from its CAN node.
- `ubyte hsk_can_msg_delete (const hsk_can_msg msg)`
Delete a CAN message object.
- `void hsk_can_msg_getData (const hsk_can_msg msg, ubyte *const msgdata)`
Gets the current data in the CAN message.
- `void hsk_can_msg_setData (const hsk_can_msg msg, const ubyte *const msgdata)`
Sets the current data in the CAN message.
- `void hsk_can_msg_send (const hsk_can_msg msg)`
Request transmission of a message.
- `bool hsk_can_msg_sent (const hsk_can_msg msg)`
Return whether the message was successfully sent between this and the previous call of this method.
- `void hsk_can_msg_receive (const hsk_can_msg msg)`
Return the message into RX mode after sending a message.
- `bool hsk_can_msg_updated (const hsk_can_msg msg)`
Return whether the message was updated via CAN bus between this call and the previous call of this method.
- `hsk_can_fifo hsk_can_fifo_create (ubyte size)`
Creates a message FIFO.
- `void hsk_can_fifo_setupRx (hsk_can_fifo fifo, const ulong id, const bool extended, const ubyte dlc)`
Set the FIFO up for receiving messages.
- `void hsk_can_fifo_setRxMask (const hsk_can_fifo fifo, ulong msk)`
Changes the ID matching mask of an RX FIFO.
- `ubyte hsk_can_fifo_move (hsk_can_fifo fifo, const ubyte list)`
Move the selected FIFO to a different list.
- `ubyte hsk_can_fifo_connect (const hsk_can_fifo fifo, const hsk_can_node node)`

- Connect a FIFO to a CAN node.*

 - ubyte [hsk_can_fifo_disconnect](#) (const [hsk_can_fifo](#) fifo)

Disconnect a FIFO from its CAN node.
- ubyte [hsk_can_fifo_delete](#) (const [hsk_can_fifo](#) fifo)

Delete a FIFO.
- void [hsk_can_fifo_next](#) (const [hsk_can_fifo](#) fifo)

Select the next FIFO entry.
- bool [hsk_can_fifo_updated](#) (const [hsk_can_fifo](#) fifo)

Return whether the currently selected FIFO entry was updated via CAN bus between this call and the previous call of this method.
- void [hsk_can_fifo_getData](#) (const [hsk_can_fifo](#) fifo, ubyte *const msgdata)

Gets the data from the currently selected FIFO entry.
- ulong [hsk_can_fifo_getId](#) (const [hsk_can_fifo](#) fifo)

Returns the CAN ID of the selected FIFO entry.
- void [hsk_can_data_setIntelSignal](#) (ubyte *const msg, ubyte bitPos, char bitCount, ulong value)

Sets a signal value in a data field.
- void [hsk_can_data_setMotorolaSignal](#) (ubyte *const msg, ubyte bitPos, char bitCount, ulong value)

Sets a big endian signal value in a data field.
- void [hsk_can_data_setSignal](#) (ubyte *const msg, const bool motorola, const bool sign, const ubyte bitPos, const char bitCount, const ulong value)

Sets a signal value in a data field.
- ulong [hsk_can_data_getIntelSignal](#) (const ubyte *const msg, const bool sign, ubyte bitPos, char bitCount)

Get a little endian signal value from a data field.
- ulong [hsk_can_data_getMotorolaSignal](#) (const ubyte *const msg, const bool sign, ubyte bitPos, char bitCount)

Get a big endian signal value from a data field.
- ulong [hsk_can_data_getSignal](#) (const ubyte *const msg, const bool motorola, const bool sign, const ubyte bitPos, const char bitCount)

Get a signal value from a data field.

Variables

- static bool [initialised](#) = 0

Stores whether common initialisation has been performed.

16.6.1 Detailed Description

HSK Controller Area Network implementation.

This file implements the functions defined in [hsk_can.h](#).

Author

kami

16.6.2 The XC878 MultitCAN Module

The following is a little excursion about CAN on the XC878.

The MultiCAN module is accessible through 3 registers:

Register	Function	Width
CAN_ADCON	CAN Address/Data Control Register	8 bits
CAN_AD	CAN Address Register	16 bits
CAN_DATA	CAN Data Register	32 bits

These registers give access to a bus. CAN_ADCON is used to control bus (e.g. write or read), everything else is done by writing the desired MultiCAN address into the CAN_AD register. The desired MultiCAN register is then accessible through the CAN_DATA register.

Register	Representation	Bits	Starting
CAN_ADCON	CAN_ADCON	8	0
CAN_AD	CAN_ADL	8	0
	CAN_ADH	8	8
	CAN_ADLH	16	0
CAN_DATA	CAN_DATA0	8	0
	CAN_DATA1	8	8
	CAN_DATA2	8	16
	CAN_DATA3	8	24
	CAN_DATA01	16	0
	CAN_DATA23	16	16

Internally the MultiCAN module has register groups, i.e. a structured set of registers that are repeated for each item having the registers. An item may be a node or a list. Each register has a fixed base address and each item a fixed offset. Each register for an item is thus addressed by setting:

```
CAN_ADLH = REGISTER + ITEM_OFFSET
```

The following example points CAN_DATA to the Node 1 Status register:

```
CAN_ADLH = NSRx + (1 << OFF_NODEx)
```

16.6.3 CAN List Management

The MultiCAN module offers 32 message objects that can be linked to one of 8 lists.

List 0 holds the unallocated (i.e. unused) objects. List 1 is connected to CAN node 0. List 2 is connected to CAN node 1.

The following implementation will use 1 of the 5 general purpose lists to park messages.

All the list management will be hidden from the "user".

16.6.4 Macro Definition Documentation

16.6.4.1 AUAD_DEC1

```
#define AUAD_DEC1 (2 << BIT_AUAD)
```

AUAD auto decrement setting.

16.6.4.2 AUAD_INC1

```
#define AUAD_INC1 (1 << BIT_AUAD)
```

AUAD auto increment setting.

16.6.4.3 AUAD_INC8

```
#define AUAD_INC8 (3 << BIT_AUAD)
```

AUAD auto increment setting.

16.6.4.4 AUAD_OFF

```
#define AUAD_OFF (0 << BIT_AUAD)
```

AUAD auto increment off setting.

16.6.4.5 BIT_ALIE

```
#define BIT_ALIE 3
```

CAN NCRx Alert Interrupt Enable bit.

16.6.4.6 BIT_AM

```
#define BIT_AM 0
```

MOAMRn Acceptance Mask for Message Identifier bits.

16.6.4.7 BIT_AUAD

```
#define BIT_AUAD 2
```

CAN_ADCON Auto Increment/Decrement the Address bits.

16.6.4.8 BIT_BRP

```
#define BIT_BRP 0
```

NBTRx Baud Rate Prescaler bits.

16.6.4.9 BIT_BSY

```
#define BIT_BSY 1
```

CAN_ADCON Data Transmission Busy bit.

16.6.4.10 BIT_BUSY

```
#define BIT_BUSY 0
```

PANCTR PANSTATUS Panel Busy Flag bit.

16.6.4.11 BIT_CALM

```
#define BIT_CALM 7
```

CAN NCRx CAN Analyze Mode bit.

16.6.4.12 BIT_CAN_DIS

```
#define BIT_CAN_DIS 5
```

PMCON1 CAN Disable Request bit.

16.6.4.13 BIT_CANDIS

```
#define BIT_CANDIS 4
```

CAN NCRx CAN Disable.

Can be used for a complete shutdown of a CAN node.

16.6.4.14 BIT_CCE

```
#define BIT_CCE 6
```

CAN NCRx Configuration Change Enable bit.

16.6.4.15 BIT_DATA

```
#define BIT_DATA 4
```

CAN_ADCON CAN Data Valid bits.

16.6.4.16 BIT_DIR

```
#define BIT_DIR 11
```

MOCTRn Direction bit.

Set this to 1 for TX, this was figured out by trial and error.

16.6.4.17 BIT_DIV8

```
#define BIT_DIV8 15
```

NBTRx Divide Prescaler Clock by 8 bit.

16.6.4.18 BIT_DLC

```
#define BIT_DLC 0
```

MOFCRn Data Length Code bits in byte 3.

Valid DLC values range from 0 to 8.

16.6.4.19 BIT_ERR

```
#define BIT_ERR 7
```

PANCTR PANAR2 Error bit.

16.6.4.20 BIT_FCCFG

```
#define BIT_FCCFG 4
```

CMCON MultiCAN Clock Configuration bit.

Used to select PCLK * 2 (1) or PCKL (0) to drive the MultiCAN module.

16.6.4.21 BIT_IDE

```
#define BIT_IDE 29
```

MOARn Identifier Extension Bit of Message Object n.

16.6.4.22 BIT_IDEXT

```
#define BIT_IDEXT 0
```

MOARn Extended CAN Identifier of Message Object n bits.

16.6.4.23 BIT_IDSTD

```
#define BIT_IDSTD 18
```

MOARn Standard CAN Identifier of Message Object n bits.

16.6.4.24 BIT_INIT

```
#define BIT_INIT 0
```

CAN NCRx Node Initialization bit.

16.6.4.25 BIT_LECIE

```
#define BIT_LECIE 2
```

CAN NCRx LEC Indicated Error Interrupt Enable bit.

16.6.4.26 BIT_LIST

```
#define BIT_LIST 4
```

MOSTATn List Allocation bits in byte 1.

16.6.4.27 BIT_MIDE

```
#define BIT_MIDE 29
```

MOAMRn Acceptance Mask Bit for Message IDE Bit.

16.6.4.28 BIT_MMC

```
#define BIT_MMC 0
```

MOFCRn Message Mode Control bits in byte 0.

16.6.4.29 BIT_MSGVAL

```
#define BIT_MSGVAL 5
```

MOCTRn/MOSTATn Message Valid bit.

16.6.4.30 BIT_NEWDAT

```
#define BIT_NEWDAT 3
```

MOCTRn/MOSTATn New Data bit.

16.6.4.31 BIT_PRI

```
#define BIT_PRI 30
```

MOARn Priority Class bits.

16.6.4.32 BIT_RBUSH

```
#define BIT_RBUSH 1
```

PANCTR PANSTATUS Result Busy Flag bit.

16.6.4.33 BIT_RWHN

```
#define BIT_RWHN 0
```

CAN_ADCON Read/Write Enable bit.

Write is 1.

16.6.4.34 BIT_RXEN

```
#define BIT_RXEN 7
```

MOCTRn/MOSTATn Receive Signal Enable bit.

16.6.4.35 BIT_RXPND

```
#define BIT_RXPND 0
```

MOCTRn/MOSTATn Receive Pending bit.

16.6.4.36 BIT_RXSEL

```
#define BIT_RXSEL 0
```

NPCRx Receive Select bit.

16.6.4.37 BIT_RXUPD

```
#define BIT_RXUPD 2
```

MOCTRn/MOSTATn Receive Updating bit.

16.6.4.38 BIT_SJW

```
#define BIT_SJW 6
```

NBTRx (Re) Synchronization Jump Width bits.

16.6.4.39 BIT_TRIE

```
#define BIT_TRIE 1
```

CAN NCRx Transfer Interrupt Enable bit.

16.6.4.40 BIT_TSEG1

```
#define BIT_TSEG1 8
```

NBTRx Time Segment Before Sample Point bits.

16.6.4.41 BIT_TSEG2

```
#define BIT_TSEG2 12
```

NBTRx Time Segment After Sample Point bits.

16.6.4.42 BIT_TXEN0

```
#define BIT_TXEN0 9
```

MOCTRn/MOSTATn Transmit Signal Enable bit.

16.6.4.43 BIT_TXEN1

```
#define BIT_TXEN1 10
```

MOCTRn/MOSTATn Transmit Signal Enable Select bit.

16.6.4.44 BIT_TXPND

```
#define BIT_TXPND 1
```

MOCTRn/MOSTATn Transmit Pending bit.

16.6.4.45 BIT_TXRQ

```
#define BIT_TXRQ 8
```

MOCTRn/MOSTATn Transmit Signal Request bit.

16.6.4.46 CAN_AD_READ

```
#define CAN_AD_READ( ) CAN_ADCON = 0
```

Sets up the CAN_AD bus for reading.

The controller always reads all 4 data bytes.

16.6.4.47 CAN_AD_READY

```
#define CAN_AD_READY( ) while (CAN_ADCON & (1 << BIT_BSY))
```

Make sure the last read/write has completed.

This is supposed to be mandatory for accessing the data bytes and CAN_ADCON, but tests show that the busy flag is never set if the module runs at 2 times PCLK, which is what this library does.

16.6.4.48 CAN_AD_WRITE

```
#define CAN_AD_WRITE(  
    msk ) CAN_ADCON = (1 << BIT_RWEN) | ((msk) << BIT_DATA)
```

Sets up the CAN_AD bus for writing.

Parameters

<i>msk</i>	A bit mask representing the data bytes that should be written. E.g. 0xC would only write CAN_DATA2 and CAN_DATA3.
------------	---

16.6.4.49 CNT_AM

```
#define CNT_AM 29
```

AM bit count.

16.6.4.50 CNT_DATA

```
#define CNT_DATA 4
```

DATA bit count.

16.6.4.51 CNT_DLC

```
#define CNT_DLC 4
```

DLC bit count.

16.6.4.52 CNT_IDEXT

```
#define CNT_IDEXT 29
```

ID bit count.

16.6.4.53 CNT_IDSTD

```
#define CNT_IDSTD 11
```

ID bit count.

16.6.4.54 CNT_LIST

```
#define CNT_LIST 4
```

LIST bit count.

16.6.4.55 CNT_MMC

```
#define CNT_MMC 4
```

MMC bit count.

16.6.4.56 CNT_PRI

```
#define CNT_PRI 2
```

PRI bit count.

16.6.4.57 CNT_RXSEL

```
#define CNT_RXSEL 3
```

RXSEL bit count.

16.6.4.58 HSK_CAN_MSG_MAX

```
#define HSK_CAN_MSG_MAX 32
```

The maximum number of message objects.

16.6.4.59 LIST_NODEx

```
#define LIST_NODEx 1
```

These lists hold message objects connected to a CAN node.

16.6.4.60 LIST_PENDING

```
#define LIST_PENDING 3
```

This list holds message objects pending assignment to a can node.

16.6.4.61 LIST_UNALLOC

```
#define LIST_UNALLOC 0
```

This list holds unallocated message objects.

16.6.4.62 MMC_DEFAULT

```
#define MMC_DEFAULT 0
```

Regular message mode.

16.6.4.63 MMC_GATEWAYSRC

```
#define MMC_GATEWAYSRC 4
```

Message is a source object for a gateway.

16.6.4.64 MMC_RXBASEFIFO

```
#define MMC_RXBASEFIFO 1
```

Message is the base of an RX FIFO.

16.6.4.65 MMC_TXBASEFIFO

```
#define MMC_TXBASEFIFO 2
```

Message is the base of a TX FIFO.

16.6.4.66 MMC_TXSLAVEFIFO

```
#define MMC_TXSLAVEFIFO 3
```

Message is a TX FIFO slave.

16.6.4.67 MOAMRn

```
#define MOAMRn 0x0403
```

Message Object n Acceptance Mask Register base address.

16.6.4.68 MOARn

```
#define MOARn 0x0406
```

Message Object n Arbitration Register base address.

16.6.4.69 MOCTRn

```
#define MOCTRn 0x0407
```

Message Object n Control Register base address.

16.6.4.70 MODATAHn

```
#define MODATAHn 0x0405
```

Message Object n Data Register High base address.

16.6.4.71 MODATALn

```
#define MODATALn 0x0404
```

Message Object n Data Register Low base address.

16.6.4.72 MOFCRn

```
#define MOFCRn 0x0400
```

Message Object n Function Control Register base address.

16.6.4.73 MOFGPRn

```
#define MOFGPRn 0x0401
```

Message Object n FIFO/Gateway Pointer Register base address.

16.6.4.74 MOFGPRn_BOT

```
#define MOFGPRn_BOT CAN_DATA0
```

MOFGPRn bottom pointer byte.

16.6.4.75 MOFGPRn_CUR

```
#define MOFGPRn_CUR CAN_DATA2
```

MOFGPRn current pointer byte.

16.6.4.76 MOFGPRn_SEL

```
#define MOFGPRn_SEL CAN_DATA3
```

MOFGPRn select pointer byte.

16.6.4.77 MOFGPRn_TOP

```
#define MOFGPRn_TOP CAN_DATA1
```

MOFGPRn top pointer byte.

16.6.4.78 MOSTATn

```
#define MOSTATn MOCTRn
```

Message Object n Status Register base address.

The status register is at the same address as the control register. It is accessed by reading from the address instead of writing.

16.6.4.79 MOSTATn_PNEXT

```
#define MOSTATn_PNEXT CAN_DATA3
```

MOSTATn Pointer to Next Message Object byte.

16.6.4.80 NBTRx

```
#define NBTRx 0x0084
```

Node x Bit Timing Register base address.

16.6.4.81 NCRx

```
#define NCRx 0x0080
```

Node x Control Register base address.

16.6.4.82 NECNTx

```
#define NECNTx 0x0085
```

Node x Error Counter Register base address.

16.6.4.83 NFCRx

```
#define NFCRx 0x0086
```

Node x Frame Counter Register base address.

16.6.4.84 NIPRx

```
#define NIPRx 0x0082
```

Node x Interrupt Pointer Register base address.

16.6.4.85 NPCRx

```
#define NPCRx 0x0083
```

Node x Port Control Register base address.

16.6.4.86 NSRx

```
#define NSRx 0x0081
```

Node x Status Register base address.

16.6.4.87 OFF_LISTm

```
#define OFF_LISTm 0
```

The Id() of the List Register (LISTm) m offset factor.

16.6.4.88 OFF_MOn

```
#define OFF_MOn 3
```

The Id() of the Message Object n offset factor.

16.6.4.89 OFF_MSIDk

```
#define OFF_MSIDk 0
```

The Id() of the Message Index Register k offset factor.

16.6.4.90 OFF_MSPNDk

```
#define OFF_MSPNDk 0
```

The Id() of the Message Pending Register k offset factor.

16.6.4.91 OFF_NODEx

```
#define OFF_NODEx 6
```

The Id() of the Node Register x offset factor.

16.6.4.92 PAN_CMD_ALLOC

```
#define PAN_CMD_ALLOC 0x03
```

List panel Dynamic Allocate command.

16.6.4.93 PAN_CMD_ALLOCBEFORE

```
#define PAN_CMD_ALLOCBEFORE 0x05
```

List panel Dynamic Insert Before command.

16.6.4.94 PAN_CMD_ALLOCBEHIND

```
#define PAN_CMD_ALLOCBEHIND 0x07
```

List panel Dynamic Insert Behind command.

16.6.4.95 PAN_CMD_INIT

```
#define PAN_CMD_INIT 0x01
```

List panel Initialize Lists command.

16.6.4.96 PAN_CMD_MOVE

```
#define PAN_CMD_MOVE 0x02
```

List panel Static Allocate command.

16.6.4.97 PAN_CMD_MOVEBEFORE

```
#define PAN_CMD_MOVEBEFORE 0x04
```

List panel Static Insert Before command.

16.6.4.98 PAN_CMD_MOVEBEHIND

```
#define PAN_CMD_MOVEBEHIND 0x06
```

List panel Static Insert Behind command.

16.6.4.99 PAN_CMD_NOP

```
#define PAN_CMD_NOP 0x00
```

List panel No Operation command.

16.6.4.100 PANAR1

```
#define PANAR1 CAN_DATA2
```

PANCTR Argument 1 Register.

16.6.4.101 PANAR2

```
#define PANAR2 CAN_DATA3
```

PANCTR Argument 2 Register.

16.6.4.102 PANCMD

```
#define PANCMD CAN_DATA0
```

PANCTR Command Register.

16.6.4.103 PANCTR

```
#define PANCTR 0x0071
```

The Panel Control Register.

All list manipulations are performed here.

16.6.4.104 PANCTR_READY

```
#define PANCTR_READY( )
```

Value:

```
do { \
    CAN_AD_READ(); \
} while (PANSTATUS & ((1 << BIT_BUSY) | (1 << BIT_RBUSY)))
```

Wait for list operations to complete.

Only execute this if CAN_ADLH points to PANCTR.

16.6.4.105 PANSTATUS

```
#define PANSTATUS CAN_DATA1
```

PANCTR Status Register.

16.6.4.106 PRI_ID

```
#define PRI_ID 2
```

CAN ID based transmit priority.

16.6.4.107 PRI_LIST

```
#define PRI_LIST 1
```

List order based transmit priority.

16.6.4.108 RESET

```
#define RESET 0x3
```

Bit mask for writing resets.

16.6.4.109 RESET_DATA

```
#define RESET_DATA CAN_DATA01
```

The register to write Control Register resets into.

16.6.4.110 SET

```
#define SET 0xC
```

Bit mask for writing settings.

16.6.4.111 SET_DATA

```
#define SET_DATA CAN_DATA23
```

The register to write Control Register settings into.

16.6.5 Function Documentation

16.6.5.1 hsk_can_data_getIntelSignal()

```
ulong hsk_can_data_getIntelSignal (  
    const ubyte *const msg,  
    const bool sign,  
    ubyte bitPos,  
    char bitCount ) [private]
```

Get a little endian signal value from a data field.

Parameters

<i>msg</i>	The message data field to read from
<i>sign</i>	Indicates whether the value has a signed type
<i>bitPos</i>	The bit position of the signal
<i>bitCount</i>	The length of the signal

Returns

The signal from the data field msg

16.6.5.2 hsk_can_data_getMotorolaSignal()

```
ulong hsk_can_data_getMotorolaSignal (  
    const ubyte *const msg,  
    const bool sign,  
    ubyte bitPos,  
    char bitCount ) [private]
```

Get a big endian signal value from a data field.

See also

[hsk_can_data_setMotorolaSignal\(\)](#) For details on the difference between big and little endian

Parameters

<i>msg</i>	The message data field to read from
<i>sign</i>	Indicates whether the value has a signed type
<i>bitPos</i>	The bit position of the signal
<i>bitCount</i>	The length of the signal

Returns

The signal from the data field msg

16.6.5.3 hsk_can_data_getSignal()

```
ulong hsk_can_data_getSignal (  
    const ubyte *const msg,  
    const bool motorola,  
    const bool sign,  
    const ubyte bitPos,  
    const char bitCount )
```

Get a signal value from a data field.

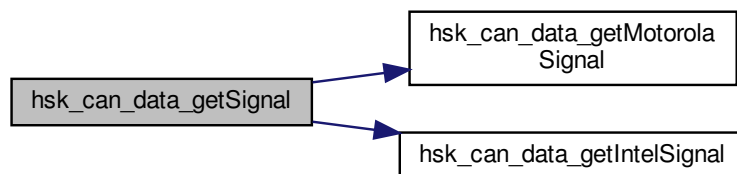
Parameters

<i>msg</i>	The message data field to read from
<i>motorola</i>	Indicates big endian (Motorola) encoding
<i>sign</i>	Indicates whether the value has a signed type
<i>bitPos</i>	The bit position of the signal
<i>bitCount</i>	The length of the signal

Returns

The signal from the data field *msg*

Here is the call graph for this function:

**16.6.5.4 hsk_can_data_setIntelSignal()**

```
void hsk_can_data_setIntelSignal (
    ubyte *const msg,
    ubyte bitPos,
    char bitCount,
    ulong value ) [private]
```

Sets a signal value in a data field.

Parameters

<i>msg</i>	The message data field to write into
<i>bitPos</i>	The bit position of the signal
<i>bitCount</i>	The length of the signal
<i>value</i>	The signal value to write into the data field

16.6.5.5 hsk_can_data_setMotorolaSignal()

```
void hsk_can_data_setMotorolaSignal (
    ubyte *const msg,
```

```

    ubyte bitPos,
    char bitCount,
    ulong value ) [private]

```

Sets a big endian signal value in a data field.

Big endian signals are bit strange, play with them in the Vector CANdb editor to figure them out.

The start position of a signal is supposed to point to the most significant bit of a signal. Consider a 10 bit message, the bits are indexed:

```

9 8 7 6 5 4 3 2 1 0

```

In that example bit 9 is the most significant bit, bit 0 the least significant. The most significant bit of a signal will be stored in the most significant bits of the message. Under the assumption that the start bit is 2, the message would be stored in the following bits:

```

Signal 9 8 7 6 5 4 3 2 1 0
Message 2 1 0 15 14 13 12 11 10 9

```

Note that the signal spreads to the most significant bits of the next byte. Special care needs to be taken, when mixing little and big endian signals. A 10 bit little endian signal with start bit 2 would cover the following message bits:

```

Signal 9 8 7 6 5 4 3 2 1 0
Message 11 10 9 8 7 6 5 4 3 2

```

Parameters

<i>msg</i>	The message data field to write into
<i>bitPos</i>	The bit position of the signal
<i>bitCount</i>	The length of the signal
<i>value</i>	The signal value to write into the data field

16.6.5.6 hsk_can_data_setSignal()

```

void hsk_can_data_setSignal (
    ubyte *const msg,
    const bool motorola,
    const bool sign,
    const ubyte bitPos,
    const char bitCount,
    const ulong value )

```

Sets a signal value in a data field.

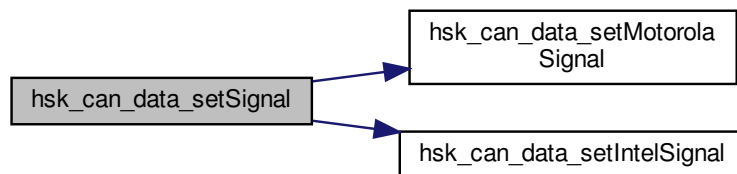
Parameters

<i>msg</i>	The message data field to write into
<i>motorola</i>	Indicates big endian (Motorola) encoding
<i>sign</i>	Indicates whether the value has a signed type

Parameters

<i>bitPos</i>	The bit position of the signal
<i>bitCount</i>	The length of the signal
<i>value</i>	The signal value to write into the data field

The sign parameter is not required for setting signals, it is just there so that one signal configuration tuple suffices for [hsk_can_data_setSignal\(\)](#) and [hsk_can_data_getSignal\(\)](#). Here is the call graph for this function:

16.6.5.7 `hsk_can_disable()`

```
void hsk_can_disable (
    const hsk_can_node node )
```

Disable a CAN node.

This completely shuts down a CAN node, cutting it off from the internal clock, to reduce energy consumption.

Parameters

<i>node</i>	The CAN node to disable
-------------	-------------------------

16.6.5.8 `hsk_can_enable()`

```
void hsk_can_enable (
    const hsk_can_node node )
```

Go live on the CAN bus.

To be called when everything is set up.

Parameters

<i>node</i>	The CAN node to enable
-------------	------------------------

16.6.5.9 hsk_can_fifo_connect()

```
ubyte hsk_can_fifo_connect (
    const hsk_can_fifo fifo,
    const hsk_can_node node )
```

Connect a FIFO to a CAN node.

Parameters

<i>fifo</i>	The identifier of the FIFO
<i>node</i>	The CAN node to connect to

Return values

<i>CAN_ERROR</i>	The given FIFO is not valid
<i>0</i>	Success

Here is the call graph for this function:



16.6.5.10 hsk_can_fifo_create()

```
hsk_can_fifo hsk_can_fifo_create (
    ubyte size )
```

Creates a message FIFO.

FIFOs can be used to ensure that multiplexed signals are not lost.

For receiving multiplexed signals it is recommended to use a FIFO as large as the number of multiplexed messages that might occur in a single burst.

If the multiplexor is large, e.g. 8 bits, it's obviously not possible to carve a 256 messages FIFO out of 32 message objects. Make an educated guess and hope that the signal provider is not hostile.

If the number of available message objects is at least one, but less than the requested length this function succeeds, but the FIFO is only created as long as possible.

Parameters

<i>size</i>	The desired FIFO size
-------------	-----------------------

Return values

<i>CAN_ERROR</i>	Creating the FIFO failed
<i>[0;32[</i>	The created FIFO id

Slave Objects

Slave objects are put into the same list as the base message object, so it can be used as a slave as well.

Always configure slave messages as TXSLAVEs, because in RXMODE the setting is ignored anyway.

Message Pointers

MOFGPRn of the base object holds the message pointers that define the list boundaries. SEL will be used to keep track of where to read/write the next message when interacting with the FIFO.

16.6.5.11 hsk_can_fifo_delete()

```
ubyte hsk_can_fifo_delete (
    const hsk_can_fifo fifo )
```

Delete a FIFO.

Parameters

<i>fifo</i>	The identifier of the FIFO
-------------	----------------------------

Return values

<i>CAN_ERROR</i>	The given FIFO is not valid
<i>0</i>	Success

Here is the call graph for this function:



16.6.5.12 hsk_can_fifo_disconnect()

```
ubyte hsk_can_fifo_disconnect (
    const hsk_can_fifo fifo )
```

Disconnect a FIFO from its CAN node.

This takes the FIFO out of active communication, without deleting it.

Parameters

<i>fifo</i>	The identifier of the FIFO
-------------	----------------------------

Return values

<i>CAN_ERROR</i>	The given FIFO is not valid
<i>0</i>	Success

Here is the call graph for this function:



16.6.5.13 hsk_can_fifo_getData()

```
void hsk_can_fifo_getData (
    const hsk_can_fifo fifo,
    ubyte *const msgdata )
```

Gets the data from the currently selected FIFO entry.

This writes DLC bytes from the FIFO entry into msgdata.

Parameters

<i>fifo</i>	The identifier of the FIFO
<i>msgdata</i>	The character array to store the message data in

Here is the call graph for this function:



16.6.5.14 hsk_can_fifo_getId()

```
ulong hsk_can_fifo_getId (
    const hsk_can_fifo fifo )
```

Returns the CAN ID of the selected FIFO entry.

Parameters

<i>fifo</i>	The ID of the FIFO
-------------	--------------------

Returns

The ID of the currently selected message object

16.6.5.15 hsk_can_fifo_move()

```
ubyte hsk_can_fifo_move (
    hsk_can_fifo fifo,
    const ubyte list ) [private]
```

Move the selected FIFO to a different list.

Parameters

<i>fifo</i>	The identifier of the FIFO
<i>list</i>	The list to move the FIFO to

Return values

<i>CAN_ERROR</i>	The given FIFO id is not valid
<i>0</i>	Move successful

16.6.5.16 hsk_can_fifo_next()

```
void hsk_can_fifo_next (
    const hsk_can_fifo fifo )
```

Select the next FIFO entry.

The [hsk_can_fifo_updated\(\)](#) and [hsk_can_fifo_getData\(\)](#) functions always refer to a certain message within the FIFO. This function selects the next entry.

Parameters

<i>fifo</i>	The ID of the FIFO to select the next entry from
-------------	--

16.6.5.17 hsk_can_fifo_setRxMask()

```
void hsk_can_fifo_setRxMask (
    const hsk_can_fifo fifo,
    ulong msk )
```

Changes the ID matching mask of an RX FIFO.

Every RX FIFO is setup to receive only on complete ID matches. This function allows updating the mask.

To generate a mask from a list of IDs use the following formula:

$$msk = \sim (id_0 | id_1 | \dots | id_n) | (id_0 \& id_1 \& \dots \& id_n)$$

Precondition

[hsk_can_fifo_setupRx\(\)](#)

Parameters

<i>fifo</i>	The FIFO to change the RX mask for
<i>msk</i>	The bit mask to set for the FIFO

16.6.5.18 hsk_can_fifo_setupRx()

```
void hsk_can_fifo_setupRx (
    hsk_can_fifo fifo,
    const ulong id,
    const bool extended,
    const ubyte dlc )
```

Set the FIFO up for receiving messages.

Parameters

<i>fifo</i>	The FIFO to setup
<i>id</i>	The message ID.
<i>extended</i>	Set this to 1 for an extended CAN message
<i>dlc</i>	The data length code, # of bytes in the message, valid values range from 0 to 8

16.6.5.19 hsk_can_fifo_updated()

```
bool hsk_can_fifo_updated (
    const hsk_can_fifo fifo )
```

Return whether the currently selected FIFO entry was updated via CAN bus between this call and the previous call of this method.

It can be used to decide when to call [hsk_can_fifo_getData\(\)](#) and [hsk_can_fifo_next\(\)](#).

Parameters

<i>fifo</i>	The identifier of the FIFO to check
-------------	-------------------------------------

Return values

1	The FIFO entry was updated since the last call of this function
0	The FIFO entry has not been updated since the last call of this function

Here is the call graph for this function:



16.6.5.20 hsk_can_init()

```

void hsk_can_init (
    const ubyte pins,
    const ulong baud )
  
```

Setup CAN communication with the desired baud rate.

The CAN node is chosen with the pin configuration.

The bus still needs to be enabled after being setup.

Parameters

<i>pins</i>	Choose one of 7 CANn_IO_* configurations
<i>baud</i>	The target baud rate to use

Configure the Bit Timing Unit

Note

Careful study of section 16.1.3 "CAN Node Control" of the [XC878 Reference Manual](#) is advised before changing the following code.

Minima and maxima are specified in ISO 11898.

One bit is separated into 3 blocks, each of which are multiples of a time quantum. The size of the time quantum (TQ) is controlled by the BRP and DIV8 bits. Because TSYNC is fixed to a single quantum, the other segments should be made up of a minimum of TQs, so TSYNC doesn't get too short (making a bit up of more TQs requires each one to be shorter at the same baud rate). However, the minimum number of TQs is 8 and some spare quanta are needed to adjust the timing between each bit transmission.

Time Slice	Value	Minimum	Encoding
TSYNC	1	Fixed	Implicite
TSEG1	8	3	7
TSEG2	3	2	2
SWJ	4	-	3

The above values provide 4 time quantumts to adjust between bits without dropping below 8 quantumts. The adjustment value is provided with the SWJ time slice.

The sample point is between TSEG1 and TSEG2, i.e. at 75%.

This means one bit requires 12 cycles. The BRP bits can be used to achieve the desired baud rate:

$$baud = 48000000/12/BRP$$

$$BRP = 48000000/12/baud$$

The encoding of BRT is also VALUE+1.

I/O Configuration

There are 7 different I/O pin configurations, four are available to node 0 and three to node 1.

See also

Section 16.1.11 "MultiCAN Port Control" of the [XC878 Reference Manual](#)

16.6.5.21 hsk_can_msg_connect()

```
ubyte hsk_can_msg_connect (
    const hsk_can_msg msg,
    const hsk_can_node node )
```

Connect a message object to a CAN node.

Parameters

<i>msg</i>	The identifier of the message object
<i>node</i>	The CAN node to connect to

Return values

<i>CAN_ERROR</i>	The given message is not valid
<i>0</i>	Success

Here is the call graph for this function:



16.6.5.22 hsk_can_msg_create()

```

hsk_can_msg hsk_can_msg_create (
    const ulong id,
    const bool extended,
    const ubyte dlc )
  
```

Creates a new CAN message.

Note that only up to 32 messages can exist at any given time.

Extended messages have 29 bit IDs and non-extended 11 bit IDs.

Parameters

<i>id</i>	The message ID.
<i>extended</i>	Set this to 1 for an extended CAN message.
<i>dlc</i>	The data length code, # of bytes in the message, valid values range from 0 to 8.

Return values

<i>CAN_ERROR</i>	Creating the message failed
<i>[0;32[</i>	A message identifier

16.6.5.23 hsk_can_msg_delete()

```

ubyte hsk_can_msg_delete (
    const hsk_can_msg msg )
  
```

Delete a CAN message object.

Parameters

<i>msg</i>	The identifier of the message object
------------	--------------------------------------

Return values

<i>CAN_ERROR</i>	The given message is not valid
0	Success

Here is the call graph for this function:



16.6.5.24 hsk_can_msg_disconnect()

```
ubyte hsk_can_msg_disconnect (  
    const hsk_can_msg msg )
```

Disconnect a CAN message object from its CAN node.

This takes a CAN message out of active communication, without deleting it.

Parameters

<i>msg</i>	The identifier of the message object
------------	--------------------------------------

Return values

<i>CAN_ERROR</i>	The given message is not valid
0	Success

Here is the call graph for this function:



16.6.5.25 hsk_can_msg_getData()

```
void hsk_can_msg_getData (
    const hsk_can_msg msg,
    ubyte *const msgdata )
```

Gets the current data in the CAN message.

This writes DLC bytes from the CAN message object into msgdata.

Parameters

<i>msg</i>	The identifier of the message object
<i>msgdata</i>	The character array to store the message data in

16.6.5.26 hsk_can_msg_move()

```
ubyte hsk_can_msg_move (
    const hsk_can_msg msg,
    const ubyte list ) [private]
```

Move the selected message and its slaves to a different list.

Parameters

<i>msg</i>	The identifier of the message object
<i>list</i>	The list to move the message object to

Return values

<i>CAN_ERROR</i>	The given message object id is not valid
<i>0</i>	Move successful

16.6.5.27 hsk_can_msg_receive()

```
void hsk_can_msg_receive (
    const hsk_can_msg msg )
```

Return the message into RX mode after sending a message.

After sending a message the messages with the same ID from other bus participants are ignored. This restores the original setting to receive messages.

Parameters

<i>msg</i>	The identifier of the message to receive
------------	--

16.6.5.28 hsk_can_msg_send()

```
void hsk_can_msg_send (
    const hsk_can_msg msg )
```

Request transmission of a message.

Parameters

<i>msg</i>	The identifier of the message to send
------------	---------------------------------------

16.6.5.29 hsk_can_msg_sent()

```
bool hsk_can_msg_sent (
    const hsk_can_msg msg )
```

Return whether the message was successfully sent between this and the previous call of this method.

Parameters

<i>msg</i>	The identifier of the message to check
------------	--

Return values

1	The message was sent since the last call of this function
0	The message has not been sent since the last call of this function

16.6.5.30 hsk_can_msg_setData()

```
void hsk_can_msg_setData (
    const hsk_can_msg msg,
    const ubyte *const msgdata )
```

Sets the current data in the CAN message.

This writes DLC bytes from msgdata to the CAN message object.

Parameters

<i>msg</i>	The identifier of the message object
<i>msgdata</i>	The character array to get the message data from

16.6.5.31 hsk_can_msg_updated()

```
bool hsk_can_msg_updated (
    const hsk_can_msg msg )
```

Return whether the message was updated via CAN bus between this call and the previous call of this method.

An update does not entail a change of message data. It just means the message was received on the CAN bus.

This is useful for cyclic message occurrence checks.

Parameters

<i>msg</i>	The identifier of the message to check
------------	--

Return values

<i>1</i>	The message was updated since the last call of this function
<i>0</i>	The message has not been updated since the last call of this function

16.6.5.32 hsk_can_status()

```
ubyte hsk_can_status (
    const hsk_can_node node,
    const ubyte field )
```

Returns a status field of a CAN node.

Parameters

<i>node</i>	The CAN node to return the status of
<i>field</i>	The status field to select

Returns

The status field state

See also

[CAN Node Status Fields](#)

16.6.6 Variable Documentation

16.6.6.1 initialised

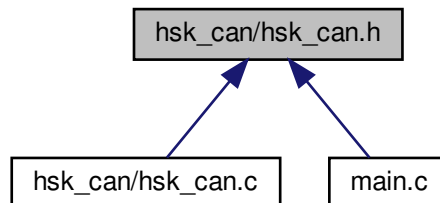
```
bool initialised = 0 [static]
```

Stores whether common initialisation has been performed.

16.7 hsk_can/hsk_can.h File Reference

HSK Controller Area Network headers.

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



Macros

- `#define CAN_ERROR 0xff`
Value returned by functions in case of an error.
- `#define CAN0 0`
CAN node 0.
- `#define CAN1 1`
CAN node 1.
- `#define CAN0_IO_P10_P11 0`
CAN node 0 IO RX on P1.0, TX on P1.1.
- `#define CAN0_IO_P16_P17 1`
CAN node 0 IO RX on P1.6, TX on P1.7.
- `#define CAN0_IO_P34_P35 2`
CAN node 0 IO RX on P3.4, TX on P3.5.
- `#define CAN0_IO_P40_P41 3`
CAN node 0 IO RX on P4.0, TX on P4.1.
- `#define CAN1_IO_P01_P02 4`
CAN node 1 IO RX on P0.1, TX on P0.2.
- `#define CAN1_IO_P14_P13 5`
CAN node 1 IO RX on P1.4, TX on P1.3.
- `#define CAN1_IO_P32_P33 6`
CAN node 1 IO RX on P3.2, TX on P3.3.
- `#define CAN_ENDIAN_INTEL 0`
Little endian signal encoding.
- `#define CAN_ENDIAN_MOTOROLA 1`
Big endian signal encoding.
- `#define CAN_STATUS_LEC 0`
The Last Error Code field provides the error triggered by the last message on the bus.
- `#define CAN_STATUS_TXOK 1`
Message Transmitted Successfully.
- `#define CAN_STATUS_RXOK 2`

- *Message Received Successfully.*
- `#define CAN_STATUS_ALERT 3`
- *Alert Warning.*
- `#define CAN_STATUS_EWRN 4`
- *Error Warning Status.*
- `#define CAN_STATUS_BOFF 5`
- *Bus-off Status.*

Typedefs

- `typedef ubyte hsk_can_node`
CAN node identifiers.
- `typedef ubyte hsk_can_msg`
CAN message object identifiers.
- `typedef ubyte hsk_can_fifo`
CAN message FIFO identifiers.

Functions

- `void hsk_can_init (const ubyte pins, const ulong baud)`
Setup CAN communication with the desired baud rate.
- `void hsk_can_enable (const hsk_can_node node)`
Go live on the CAN bus.
- `void hsk_can_disable (const hsk_can_node node)`
Disable a CAN node.
- `ubyte hsk_can_status (const hsk_can_node node, const ubyte field)`
Returns a status field of a CAN node.
- `hsk_can_msg hsk_can_msg_create (const ulong id, const bool extended, const ubyte dlc)`
Creates a new CAN message.
- `ubyte hsk_can_msg_connect (const hsk_can_msg msg, const hsk_can_node node)`
Connect a message object to a CAN node.
- `ubyte hsk_can_msg_disconnect (const hsk_can_msg msg)`
Disconnect a CAN message object from its CAN node.
- `ubyte hsk_can_msg_delete (const hsk_can_msg msg)`
Delete a CAN message object.
- `void hsk_can_msg_getData (const hsk_can_msg msg, ubyte *const msgdata)`
Gets the current data in the CAN message.
- `void hsk_can_msg_setData (const hsk_can_msg msg, const ubyte *const msgdata)`
Sets the current data in the CAN message.
- `void hsk_can_msg_send (const hsk_can_msg msg)`
Request transmission of a message.
- `bool hsk_can_msg_sent (const hsk_can_msg msg)`
Return whether the message was successfully sent between this and the previous call of this method.
- `void hsk_can_msg_receive (const hsk_can_msg msg)`
Return the message into RX mode after sending a message.
- `bool hsk_can_msg_updated (const hsk_can_msg msg)`
Return whether the message was updated via CAN bus between this call and the previous call of this method.
- `hsk_can_fifo hsk_can_fifo_create (ubyte size)`
Creates a message FIFO.

- void [hsk_can_fifo_setupRx](#) ([hsk_can_fifo](#) fifo, const ulong id, const bool extended, const ubyte dlc)
Set the FIFO up for receiving messages.
- void [hsk_can_fifo_setRxMask](#) (const [hsk_can_fifo](#) fifo, ulong msk)
Changes the ID matching mask of an RX FIFO.
- ubyte [hsk_can_fifo_connect](#) (const [hsk_can_fifo](#) fifo, const [hsk_can_node](#) node)
Connect a FIFO to a CAN node.
- ubyte [hsk_can_fifo_disconnect](#) (const [hsk_can_fifo](#) fifo)
Disconnect a FIFO from its CAN node.
- ubyte [hsk_can_fifo_delete](#) (const [hsk_can_fifo](#) fifo)
Delete a FIFO.
- void [hsk_can_fifo_next](#) (const [hsk_can_fifo](#) fifo)
Select the next FIFO entry.
- ulong [hsk_can_fifo_getId](#) (const [hsk_can_fifo](#) fifo)
Returns the CAN ID of the selected FIFO entry.
- bool [hsk_can_fifo_updated](#) (const [hsk_can_fifo](#) fifo)
Return whether the currently selected FIFO entry was updated via CAN bus between this call and the previous call of this method.
- void [hsk_can_fifo_getData](#) (const [hsk_can_fifo](#) fifo, ubyte *const msgdata)
Gets the data from the currently selected FIFO entry.
- void [hsk_can_data_setSignal](#) (ubyte *const msg, const bool motorola, const bool sign, const ubyte bitPos, const char bitCount, const ulong value)
Sets a signal value in a data field.
- ulong [hsk_can_data_getSignal](#) (const ubyte *const msg, const bool motorola, const bool sign, const ubyte bitPos, const char bitCount)
Get a signal value from a data field.

16.7.1 Detailed Description

HSK Controller Area Network headers.

This file contains the function prototypes to initialize and engage in CAN communication over the builtin CAN nodes 0 and 1.

Author

kami

16.7.2 CAN Message/Signal Tuples

The recommended way to use messages and signals is not to specify them inline, but to provide defines with a set of parameters.

These tuples should follow the following pattern:

```
#define MSG_<MSGNAME>      <id>, <extended>, <dlc>
#define SIG_<SIGNAME>      <motorola>, <signed>, <bitPos>, <bitCount>
```

The symbols have the following meaning:

- MSGNAME: The name of the message in capitals, e.g. AFB_CHANNELS

- id: The CAN id of the message, e.g. 0x403
- extended: Whether the CAN ID is extended or not, e.g. 0 for a regular ID
- dlc: The data length count of the message, e.g. 3
- SIGNAME: The name of the signal in capitals, e.g. AFB_CHANNEL0_CURRENT
- motorola: Whether the signal is in big endian (Motorola) format
- signed: Whether the signal is signed
- bitPos: The starting bit of the signal, e.g. 0
- bitCount: The length of the signal in bits, e.g. 10

Tuples using the specified format can directly be used as parameters for several functions in the library.

16.7.3 CAN Node Management

There are 7 port pairs available for CAN communication, check the `CANn_IO_*` defines. Four for the node CAN0 and three for CAN1.

16.7.4 Message Object Management

The MultiCAN module offers up to 32 message objects. New messages are set up for receiving messages. Message object can be switched from RX to TX mode and back with the `hsk_can_msg_send()` and `hsk_can_msg_receive()` functions.

16.7.5 FIFOs

FIFOs are the weapon of choice when dealing with large numbers of individual messages or when receiving multiplexed data. In most use cases only the latest version of a message is relevant and FIFOs are not required. But messages containing multiplexed signals may contain critical signals that would be overwritten by a message with the same ID, but a different multiplexor.

If more message IDs than available message objects are used to send and/or receive data, there is no choice but to use a FIFO.

Currently only RX FIFOs are supported.

A FIFO can act as a buffer the CAN module can store message data in until it can be dealt with. The following example illustrates how to read from a FIFO:

```
if (hsk_can_fifo_updated(fifo0)) {
    hsk_can_fifo_getData(fifo0, data0);
    hsk_can_fifo_next(fifo0);
    select = hsk_can_data_getSignal(data0, SIG_MULTIPLEXOR);
    [...]
}
```

When using a mask to accept several messages checking the ID becomes necessary:

```
if (hsk_can_fifo_updated(fifo0)) {
    switch (hsk_can_fifo_getId()) {
        case MSG_0_ID:
            hsk_can_fifo_getData(fifo0, data0);
            [...]
            break;
        case MSG_1_ID:
            hsk_can_fifo_getData(fifo0, data1);
            [...]
            break;
        [...]
    }
    hsk_can_fifo_next(fifo0);
}
```

FIFOs draw from the same message object pool regular message objects do.

16.7.6 Message Data

The [hsk_can_data_setSignal\(\)](#) and [hsk_can_data_getSignal\(\)](#) functions allow writing and reading signals across byte boundaries to and from a buffer.

For big endian signals the bit position of the most significant bit must be supplied (highest bit in the first byte). For little endian signals the least significant bit must be supplied (lowest bit in the first byte).

This conforms to the way signal positions are stored in Vector CANdb++ DBC files.

16.7.7 Macro Definition Documentation

16.7.7.1 CAN0

```
#define CAN0 0
```

CAN node 0.

16.7.7.2 CAN0_IO_P10_P11

```
#define CAN0_IO_P10_P11 0
```

CAN node 0 IO RX on P1.0, TX on P1.1.

16.7.7.3 CAN0_IO_P16_P17

```
#define CAN0_IO_P16_P17 1
```

CAN node 0 IO RX on P1.6, TX on P1.7.

16.7.7.4 CAN0_IO_P34_P35

```
#define CAN0_IO_P34_P35 2
```

CAN node 0 IO RX on P3.4, TX on P3.5.

16.7.7.5 CAN0_IO_P40_P41

```
#define CAN0_IO_P40_P41 3
```

CAN node 0 IO RX on P4.0, TX on P4.1.

16.7.7.6 CAN1

```
#define CAN1 1
```

CAN node 1.

16.7.7.7 CAN1_IO_P01_P02

```
#define CAN1_IO_P01_P02 4
```

CAN node 1 IO RX on P0.1, TX on P0.2.

16.7.7.8 CAN1_IO_P14_P13

```
#define CAN1_IO_P14_P13 5
```

CAN node 1 IO RX on P1.4, TX on P1.3.

16.7.7.9 CAN1_IO_P32_P33

```
#define CAN1_IO_P32_P33 6
```

CAN node 1 IO RX on P3.2, TX on P3.3.

16.7.7.10 CAN_ENDIAN_INTEL

```
#define CAN_ENDIAN_INTEL 0
```

Little endian signal encoding.

Deprecated In favour of shorter and cleaner code the [hsk_can_data_getSignal\(\)](#) and [hsk_can_data_setSignal\(\)](#) functions were switched to using boolean (motorola positive) logic

16.7.7.11 CAN_ENDIAN_MOTOROLA

```
#define CAN_ENDIAN_MOTOROLA 1
```

Big endian signal encoding.

Deprecated In favour of shorter and cleaner code the [hsk_can_data_getSignal\(\)](#) and [hsk_can_data_setSignal\(\)](#) functions were switched to using boolean (motorola positive) logic

16.7.7.12 CAN_ERROR

```
#define CAN_ERROR 0xff
```

Value returned by functions in case of an error.

16.7.8 Typedef Documentation

16.7.8.1 hsk_can_fifo

```
typedef ubyte hsk_can_fifo
```

CAN message FIFO identifiers.

16.7.8.2 hsk_can_msg

```
typedef ubyte hsk_can_msg
```

CAN message object identifiers.

16.7.8.3 hsk_can_node

```
typedef ubyte hsk_can_node
```

CAN node identifiers.

16.7.9 Function Documentation

16.7.9.1 hsk_can_data_getSignal()

```
ulong hsk_can_data_getSignal (
    const ubyte *const msg,
    const bool  motorola,
    const bool  sign,
    const ubyte bitPos,
    const char  bitCount )
```

Get a signal value from a data field.

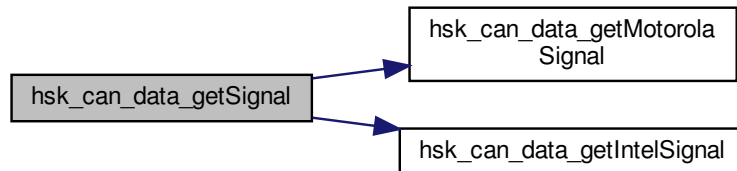
Parameters

<i>msg</i>	The message data field to read from
<i>motorola</i>	Indicates big endian (Motorola) encoding
<i>sign</i>	Indicates whether the value has a signed type
<i>bitPos</i>	The bit position of the signal
<i>bitCount</i>	The length of the signal

Returns

The signal from the data field msg

Here is the call graph for this function:

**16.7.9.2 hsk_can_data_setSignal()**

```

void hsk_can_data_setSignal (
    ubyte *const msg,
    const bool motorola,
    const bool sign,
    const ubyte bitPos,
    const char bitCount,
    const ulong value )
  
```

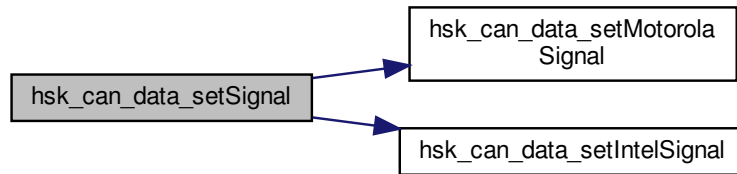
Sets a signal value in a data field.

Parameters

<i>msg</i>	The message data field to write into
<i>motorola</i>	Indicates big endian (Motorola) encoding
<i>sign</i>	Indicates whether the value has a signed type
<i>bitPos</i>	The bit position of the signal
<i>bitCount</i>	The length of the signal
<i>value</i>	The signal value to write into the data field

The sign parameter is not required for setting signals, it is just there so that one signal configuration tuple suffices

for [hsk_can_data_setSignal\(\)](#) and [hsk_can_data_getSignal\(\)](#). Here is the call graph for this function:



16.7.9.3 `hsk_can_disable()`

```
void hsk_can_disable (
    const hsk\_can\_node node )
```

Disable a CAN node.

This completely shuts down a CAN node, cutting it off from the internal clock, to reduce energy consumption.

Parameters

<i>node</i>	The CAN node to disable
-------------	-------------------------

16.7.9.4 `hsk_can_enable()`

```
void hsk_can_enable (
    const hsk\_can\_node node )
```

Go live on the CAN bus.

To be called when everything is set up.

Parameters

<i>node</i>	The CAN node to enable
-------------	------------------------

16.7.9.5 `hsk_can_fifo_connect()`

```
ubyte hsk_can_fifo_connect (
    const hsk\_can\_fifo fifo,
    const hsk\_can\_node node )
```

Connect a FIFO to a CAN node.

Parameters

<i>fifo</i>	The identifier of the FIFO
<i>node</i>	The CAN node to connect to

Return values

<i>CAN_ERROR</i>	The given FIFO is not valid
0	Success

Here is the call graph for this function:



16.7.9.6 hsk_can_fifo_create()

```

hsk_can_fifo hsk_can_fifo_create (
    ubyte size )
  
```

Creates a message FIFO.

FIFOs can be used to ensure that multiplexed signals are not lost.

For receiving multiplexed signals it is recommended to use a FIFO as large as the number of multiplexed messages that might occur in a single burst.

If the multiplexor is large, e.g. 8 bits, it's obviously not possible to carve a 256 messages FIFO out of 32 message objects. Make an educated guess and hope that the signal provider is not hostile.

If the number of available message objects is at least one, but less than the requested length this function succeeds, but the FIFO is only created as long as possible.

Parameters

<i>size</i>	The desired FIFO size
-------------	-----------------------

Return values

<i>CAN_ERROR</i>	Creating the FIFO failed
[0;32[The created FIFO id

Slave Objects

Slave objects are put into the same list as the base message object, so it can be used as a slave as well.

Always configure slave messages as TXSLAVEs, because in RXMODE the setting is ignored anyway.

Message Pointers

MOFGPRn of the base object holds the message pointers that define the list boundaries. SEL will be used to keep track of where to read/write the next message when interacting with the FIFO.

16.7.9.7 hsk_can_fifo_delete()

```
ubyte hsk_can_fifo_delete (
    const hsk_can_fifo fifo )
```

Delete a FIFO.

Parameters

<i>fifo</i>	The identifier of the FIFO
-------------	----------------------------

Return values

<i>CAN_ERROR</i>	The given FIFO is not valid
<i>0</i>	Success

Here is the call graph for this function:



16.7.9.8 hsk_can_fifo_disconnect()

```
ubyte hsk_can_fifo_disconnect (
    const hsk_can_fifo fifo )
```

Disconnect a FIFO from its CAN node.

This takes the FIFO out of active communication, without deleting it.

Parameters

<i>fifo</i>	The identifier of the FIFO
-------------	----------------------------

Return values

<i>CAN_ERROR</i>	The given FIFO is not valid
<i>0</i>	Success

Here is the call graph for this function:



16.7.9.9 hsk_can_fifo_getData()

```

void hsk_can_fifo_getData (
    const hsk_can_fifo fifo,
    ubyte *const msgdata )
  
```

Gets the data from the currently selected FIFO entry.

This writes DLC bytes from the FIFO entry into msgdata.

Parameters

<i>fifo</i>	The identifier of the FIFO
<i>msgdata</i>	The character array to store the message data in

Here is the call graph for this function:



16.7.9.10 hsk_can_fifo_getId()

```

ulong hsk_can_fifo_getId (
    const hsk_can_fifo fifo )
  
```

Returns the CAN ID of the selected FIFO entry.

Parameters

<i>fifo</i>	The ID of the FIFO
-------------	--------------------

Returns

The ID of the currently selected message object

16.7.9.11 hsk_can_fifo_next()

```
void hsk_can_fifo_next (
    const hsk_can_fifo fifo )
```

Select the next FIFO entry.

The [hsk_can_fifo_updated\(\)](#) and [hsk_can_fifo_getData\(\)](#) functions always refer to a certain message within the FIFO. This function selects the next entry.

Parameters

<i>fifo</i>	The ID of the FIFO to select the next entry from
-------------	--

16.7.9.12 hsk_can_fifo_setRxMask()

```
void hsk_can_fifo_setRxMask (
    const hsk_can_fifo fifo,
    ulong msk )
```

Changes the ID matching mask of an RX FIFO.

Every RX FIFO is setup to receive only on complete ID matches. This function allows updating the mask.

To generate a mask from a list of IDs use the following formula:

$$msk = \sim (id_0 | id_1 | \dots | id_n) | (id_0 \& id_1 \& \dots \& id_n)$$

Precondition

[hsk_can_fifo_setupRx\(\)](#)

Parameters

<i>fifo</i>	The FIFO to change the RX mask for
<i>msk</i>	The bit mask to set for the FIFO

16.7.9.13 hsk_can_fifo_setupRx()

```
void hsk_can_fifo_setupRx (
    hsk_can_fifo fifo,
    const ulong id,
    const bool extended,
    const ubyte dlc )
```

Set the FIFO up for receiving messages.

Parameters

<i>fifo</i>	The FIFO to setup
<i>id</i>	The message ID.
<i>extended</i>	Set this to 1 for an extended CAN message
<i>dlc</i>	The data length code, # of bytes in the message, valid values range from 0 to 8

16.7.9.14 hsk_can_fifo_updated()

```
bool hsk_can_fifo_updated (
    const hsk_can_fifo fifo )
```

Return whether the currently selected FIFO entry was updated via CAN bus between this call and the previous call of this method.

It can be used to decide when to call [hsk_can_fifo_getData\(\)](#) and [hsk_can_fifo_next\(\)](#).

Parameters

<i>fifo</i>	The identifier of the FIFO to check
-------------	-------------------------------------

Return values

1	The FIFO entry was updated since the last call of this function
0	The FIFO entry has not been updated since the last call of this function

Here is the call graph for this function:



16.7.9.15 hsk_can_init()

```
void hsk_can_init (
    const ubyte pins,
    const ulong baud )
```

Setup CAN communication with the desired baud rate.

The CAN node is chosen with the pin configuration.

The bus still needs to be enabled after being setup.

Parameters

<i>pins</i>	Choose one of 7 CANn_IO_* configurations
<i>baud</i>	The target baud rate to use

Configure the Bit Timing Unit

Note

Careful study of section 16.1.3 "CAN Node Control" of the [XC878 Reference Manual](#) is advised before changing the following code.

Minima and maxima are specified in ISO 11898.

One bit is separated into 3 blocks, each of which are multiples of a time quantum. The size of the time quantum (TQ) is controlled by the BRP and DIV8 bits. Because TSYNC is fixed to a single quantum, the other segments should be made up of a minimum of TQs, so TSYNC doesn't get too short (making a bit up of more TQs requires each one to be shorter at the same baud rate). However, the minimum number of TQs is 8 and some spare quanta are needed to adjust the timing between each bit transmission.

Time Slice	Value	Minimum	Encoding
TSYNC	1	Fixed	Implicite
TSEG1	8	3	7
TSEG2	3	2	2
SWJ	4	-	3

The above values provide 4 time quanta to adjust between bits without dropping below 8 quanta. The adjustment value is provided with the SWJ time slice.

The sample point is between TSEG1 and TSEG2, i.e. at 75%.

This means one bit requires 12 cycles. The BRP bits can be used to achieve the desired baud rate:

$$baud = 48000000/12/BRP$$

$$BRP = 48000000/12/baud$$

The encoding of BRT is also VALUE+1.

I/O Configuration

There are 7 different I/O pin configurations, four are available to node 0 and three to node 1.

See also

Section 16.1.11 "MultiCAN Port Control" of the [XC878 Reference Manual](#)

16.7.9.16 hsk_can_msg_connect()

```
ubyte hsk_can_msg_connect (
    const hsk_can_msg msg,
    const hsk_can_node node )
```

Connect a message object to a CAN node.

Parameters

<i>msg</i>	The identifier of the message object
<i>node</i>	The CAN node to connect to

Return values

<i>CAN_ERROR</i>	The given message is not valid
<i>0</i>	Success

Here is the call graph for this function:



16.7.9.17 hsk_can_msg_create()

```
hsk_can_msg hsk_can_msg_create (
    const ulong id,
    const bool extended,
    const ubyte dlc )
```

Creates a new CAN message.

Note that only up to 32 messages can exist at any given time.

Extended messages have 29 bit IDs and non-extended 11 bit IDs.

Parameters

<i>id</i>	The message ID.
<i>extended</i>	Set this to 1 for an extended CAN message.
<i>dlc</i>	The data length code, # of bytes in the message, valid values range from 0 to 8.

Return values

<i>CAN_ERROR</i>	Creating the message failed
<i>[0;32[</i>	A message identifier

16.7.9.18 hsk_can_msg_delete()

```
ubyte hsk_can_msg_delete (
    const hsk_can_msg msg )
```

Delete a CAN message object.

Parameters

<i>msg</i>	The identifier of the message object
------------	--------------------------------------

Return values

<i>CAN_ERROR</i>	The given message is not valid
<i>0</i>	Success

Here is the call graph for this function:



16.7.9.19 hsk_can_msg_disconnect()

```
ubyte hsk_can_msg_disconnect (
    const hsk_can_msg msg )
```

Disconnect a CAN message object from its CAN node.

This takes a CAN message out of active communication, without deleting it.

Parameters

<i>msg</i>	The identifier of the message object
------------	--------------------------------------

Return values

<i>CAN_ERROR</i>	The given message is not valid
<i>0</i>	Success

Here is the call graph for this function:



16.7.9.20 hsk_can_msg_getData()

```

void hsk_can_msg_getData (
    const hsk_can_msg msg,
    ubyte *const msgdata )
  
```

Gets the current data in the CAN message.

This writes DLC bytes from the CAN message object into msgdata.

Parameters

<i>msg</i>	The identifier of the message object
<i>msgdata</i>	The character array to store the message data in

16.7.9.21 hsk_can_msg_receive()

```

void hsk_can_msg_receive (
    const hsk_can_msg msg )
  
```

Return the message into RX mode after sending a message.

After sending a message the messages with the same ID from other bus participants are ignored. This restores the original setting to receive messages.

Parameters

<i>msg</i>	The identifier of the message to receive
------------	--

16.7.9.22 hsk_can_msg_send()

```
void hsk_can_msg_send (
    const hsk_can_msg msg )
```

Request transmission of a message.

Parameters

<i>msg</i>	The identifier of the message to send
------------	---------------------------------------

16.7.9.23 hsk_can_msg_sent()

```
bool hsk_can_msg_sent (
    const hsk_can_msg msg )
```

Return whether the message was successfully sent between this and the previous call of this method.

Parameters

<i>msg</i>	The identifier of the message to check
------------	--

Return values

1	The message was sent since the last call of this function
0	The message has not been sent since the last call of this function

16.7.9.24 hsk_can_msg_setData()

```
void hsk_can_msg_setData (
    const hsk_can_msg msg,
    const ubyte *const msgdata )
```

Sets the current data in the CAN message.

This writes DLC bytes from msgdata to the CAN message object.

Parameters

<i>msg</i>	The identifier of the message object
<i>msgdata</i>	The character array to get the message data from

16.7.9.25 hsk_can_msg_updated()

```
bool hsk_can_msg_updated (
    const hsk_can_msg msg )
```

Return whether the message was updated via CAN bus between this call and the previous call of this method.

An update does not entail a change of message data. It just means the message was received on the CAN bus.

This is useful for cyclic message occurrence checks.

Parameters

<i>msg</i>	The identifier of the message to check
------------	--

Return values

1	The message was updated since the last call of this function
0	The message has not been updated since the last call of this function

16.7.9.26 hsk_can_status()

```
ubyte hsk_can_status (
    const hsk_can_node node,
    const ubyte field )
```

Returns a status field of a CAN node.

Parameters

<i>node</i>	The CAN node to return the status of
<i>field</i>	The status field to select

Returns

The status field state

See also

[CAN Node Status Fields](#)

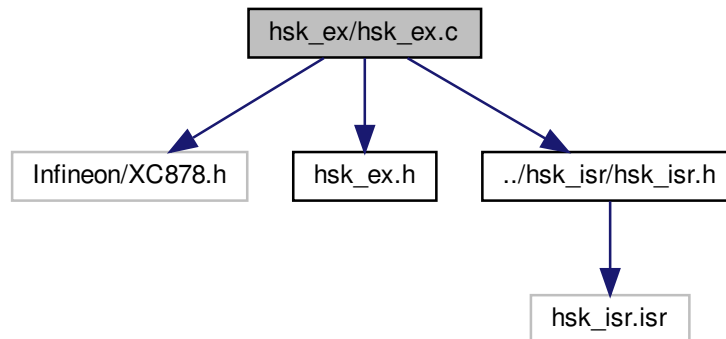
16.8 hsk_ex/hsk_ex.c File Reference

HSK External Interrupt Routing implementation.

```
#include <Infineon/XC878.h>
#include "hsk_ex.h"
```

```
#include "../hsk_isr/hsk_isr.h"
```

Include dependency graph for hsk_ex.c:



Macros

- `#define CNT_EXINT 2`
EXICON0/1 External Interrupt Trigger Select bit count.
- `#define BIT_EXINT0 0`
EXICON0 External Interrupt 0 Trigger Select bits.
- `#define BIT_EXINT1 2`
EXICON0 External Interrupt 1 Trigger Select bits.
- `#define BIT_EXINT2 4`
EXICON0 External Interrupt 2 Trigger Select bits.
- `#define BIT_EXINT3 6`
EXICON0 External Interrupt 3 Trigger Select bits.
- `#define BIT_EXINT4 0`
EXICON1 External Interrupt 4 Trigger Select bits.
- `#define BIT_EXINT5 2`
EXICON1 External Interrupt 5 Trigger Select bits.
- `#define BIT_EXINT6 4`
EXICON1 External Interrupt 6 Trigger Select bits.
- `#define BIT_IMODE 4`
SYSCON0 Interrupt Structure 2 Mode Select bit.
- `#define EX_EDGE_DISABLE 3`
Deactivate external interrupt.

Functions

- `void hsk_ex_channel_enable` (const `hsk_ex_channel` channel, const `ubyte` edge, const `void(*const callback)(void)`)
Enable an external interrupt channel.
- `void hsk_ex_channel_disable` (const `hsk_ex_channel` channel)
Disables an external interrupt channel.
- `void hsk_ex_port_open` (const `hsk_ex_port` port)
Opens an input port for an external interrupt.
- `void hsk_ex_port_close` (const `hsk_ex_port` port)
Disconnects an input port from an external interrupt.

Variables

- struct {
 - ubyte [modpiselBit](#)
The *MODPISEL[n]* bit(s) to select.
 - ubyte [modpiselSel](#)
The *MODPISEL* value.
 - ubyte [portBit](#)
The *port* bit.
 - ubyte [portAltSel](#)
The *port ALTSEL* (alternative select) setting.
- } [ports](#) []

External input configuration structure.

16.8.1 Detailed Description

HSK External Interrupt Routing implementation.

This file implements the methods necessary to route μ C pins to external interrupts.

Author

kami

16.8.2 Macro Definition Documentation

16.8.2.1 BIT_EXINT0

```
#define BIT_EXINT0 0
```

EXICON0 External Interrupt 0 Trigger Select bits.

16.8.2.2 BIT_EXINT1

```
#define BIT_EXINT1 2
```

EXICON0 External Interrupt 1 Trigger Select bits.

16.8.2.3 BIT_EXINT2

```
#define BIT_EXINT2 4
```

EXICON0 External Interrupt 2 Trigger Select bits.

16.8.2.4 BIT_EXINT3

```
#define BIT_EXINT3 6
```

EXICON0 External Interrupt 3 Trigger Select bits.

16.8.2.5 BIT_EXINT4

```
#define BIT_EXINT4 0
```

EXICON1 External Interrupt 4 Trigger Select bits.

16.8.2.6 BIT_EXINT5

```
#define BIT_EXINT5 2
```

EXICON1 External Interrupt 5 Trigger Select bits.

16.8.2.7 BIT_EXINT6

```
#define BIT_EXINT6 4
```

EXICON1 External Interrupt 6 Trigger Select bits.

16.8.2.8 BIT_IMODE

```
#define BIT_IMODE 4
```

SYSCON0 Interrupt Structure 2 Mode Select bit.

16.8.2.9 CNT_EXINT

```
#define CNT_EXINT 2
```

EXICON0/1 External Interrupt Trigger Select bit count.

16.8.3 Function Documentation

16.8.3.1 hsk_ex_channel_disable()

```
void hsk_ex_channel_disable (
    const hsk_ex_channel channel )
```

Disables an external interrupt channel.

Parameters

<i>channel</i>	The channel to disable, one of External Interrupt Channels
----------------	--

16.8.3.2 hsk_ex_channel_enable()

```
void hsk_ex_channel_enable (
    const hsk_ex_channel channel,
    const ubyte edge,
    const void(*) (void) callback )
```

Enable an external interrupt channel.

It is good practice to enable a port for the channel first, because port changes on an active interrupt may cause an undesired interrupt.

The callback function can be set to 0 if a change of the function is not desired. For channels EXINT0 and EXINT1 the callback is ignored, implement interrupts 0 and 2 instead.

Parameters

<i>channel</i>	The channel to activate, one of External Interrupt Channels
<i>edge</i>	The triggering edge, one of External Interrupt Triggers
<i>callback</i>	The callback function for an interrupt event

Setting up EXINT0/1 is somewhat confusing. Refer to UM 1.1 section 5.6.2 to make sense of this.

16.8.3.3 hsk_ex_port_close()

```
void hsk_ex_port_close (
    const hsk_ex_port port )
```

Disconnects an input port from an external interrupt.

Parameters

<i>port</i>	The port to close, one of External Interrupt Input Ports
-------------	--

16.8.3.4 hsk_ex_port_open()

```
void hsk_ex_port_open (
    const hsk_ex_port port )
```

Opens an input port for an external interrupt.

Parameters

<i>port</i>	The port to open, one of External Interrupt Input Ports
-------------	---

16.8.4 Variable Documentation

16.8.4.1 modpiselBit

```
ubyte modpiselBit
```

The MODPISEL[n] bit(s) to select.

16.8.4.2 modpiselSel

```
ubyte modpiselSel
```

The MODPISEL value.

16.8.4.3 portAltsel

```
ubyte portAltsel
```

The port ALTSEL (alternative select) setting.

16.8.4.4 portBit

```
ubyte portBit
```

The port bit.

16.8.4.5 ports

```
ports [static]
```

Initial value:

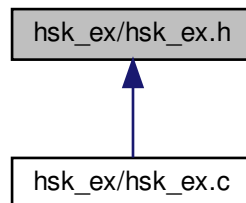
```
= {
    {1, 0, 5, 2},
    {0, 0, 1, 2},
    {1, 1, 4, 2},
    {4, 0, 5, 2},
    {5, 0, 6, 3},
    {0, 2, 0, 3},
    {2, 2, 2, 5},
    {4, 2, 3, 1},
    {5, 3, 4, 4},
    {2, 0, 7, 2},
    {0, 1, 0, 4},
    {2, 2, 1, 1},
    {5, 1, 2, 2},
    {4, 1, 4, 3},
    {5, 2, 5, 3},
    {2, 1, 0, 2},
    {3, 1, 1, 2},
    {4, 3, 2, 2},
    {2, 0, 3, 2},
    {3, 0, 4, 2},
    {0, 3, 5, 2},
    {2, 3, 6, 2},
    {5, 4, 7, 3}
}
```

External input configuration structure.

16.9 hsk_ex/hsk_ex.h File Reference

HSK External Interrupt Routing headers.

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



Macros

- `#define EX_EXINT0 0`
External interrupt channel EXINT0.
- `#define EX_EXINT1 1`
External interrupt channel EXINT1.
- `#define EX_EXINT2 2`
External interrupt channel EXINT2.
- `#define EX_EXINT3 3`
External interrupt channel EXINT3.
- `#define EX_EXINT4 4`
External interrupt channel EXINT4.
- `#define EX_EXINT5 5`
External interrupt channel EXINT5.
- `#define EX_EXINT6 6`
External interrupt channel EXINT6.
- `#define EX_EDGE_RISING 0`
Trigger interrupt on rising edge.
- `#define EX_EDGE_FALLING 1`
Trigger interrupt on falling edge.
- `#define EX_EDGE_BOTH 2`
Trigger interrupt on both edges.
- `#define EX_EXINT0_P05 0`
External interrupt EXINT0 input port P0.5.
- `#define EX_EXINT3_P11 1`
External interrupt EXINT3 input port P1.1.
- `#define EX_EXINT0_P14 2`
External interrupt EXINT0 input port P1.4.
- `#define EX_EXINT5_P15 3`
External interrupt EXINT5 input port P1.5.
- `#define EX_EXINT6_P16 4`

- External interrupt EXINT6 input port P1.6.*

 - #define [EX_EXINT3_P30](#) 5

External interrupt EXINT3 input port P3.0.

 - #define [EX_EXINT4_P32](#) 6

External interrupt EXINT4 input port P3.2.

 - #define [EX_EXINT5_P33](#) 7

External interrupt EXINT5 input port P3.3.

 - #define [EX_EXINT6_P34](#) 8

External interrupt EXINT6 input port P3.4.

 - #define [EX_EXINT4_P37](#) 9

External interrupt EXINT4 input port P3.7.

 - #define [EX_EXINT3_P40](#) 10

External interrupt EXINT3 input port P4.0.

 - #define [EX_EXINT4_P41](#) 11

External interrupt EXINT4 input port P4.1.

 - #define [EX_EXINT6_P42](#) 12

External interrupt EXINT6 input port P4.2.

 - #define [EX_EXINT5_P44](#) 13

External interrupt EXINT5 input port P4.4.

 - #define [EX_EXINT6_P45](#) 14

External interrupt EXINT6 input port P4.5.

 - #define [EX_EXINT1_P50](#) 15

External interrupt EXINT1 input port P5.0.

 - #define [EX_EXINT2_P51](#) 16

External interrupt EXINT2 input port P5.1.

 - #define [EX_EXINT5_P52](#) 17

External interrupt EXINT5 input port P5.2.

 - #define [EX_EXINT1_P53](#) 18

External interrupt EXINT1 input port P5.3.

 - #define [EX_EXINT2_P54](#) 19

External interrupt EXINT2 input port P5.4.

 - #define [EX_EXINT3_P55](#) 20

External interrupt EXINT3 input port P5.5.

 - #define [EX_EXINT4_P56](#) 21

External interrupt EXINT4 input port P5.6.

 - #define [EX_EXINT6_P57](#) 22

External interrupt EXINT6 input port P5.7.

Typedefs

- typedef ubyte [hsk_ex_channel](#)

Typedef for external interrupt channels.
- typedef ubyte [hsk_ex_port](#)

Typedef for external interrupt ports.

Functions

- void [hsk_ex_channel_enable](#) (const [hsk_ex_channel](#) channel, const ubyte edge, const void(*const [call-back](#))(void))
Enable an external interrupt channel.
- void [hsk_ex_channel_disable](#) (const [hsk_ex_channel](#) channel)
Disables an external interrupt channel.
- void [hsk_ex_port_open](#) (const [hsk_ex_port](#) port)
Opens an input port for an external interrupt.
- void [hsk_ex_port_close](#) (const [hsk_ex_port](#) port)
Disconnects an input port from an external interrupt.

16.9.1 Detailed Description

HSK External Interrupt Routing headers.

This file offers functions to activate external interrupts and connect them to the available input pins.

Author

kami

16.9.2 Typedef Documentation

16.9.2.1 [hsk_ex_channel](#)

```
typedef ubyte hsk\_ex\_channel
```

Typedef for external interrupt channels.

16.9.2.2 [hsk_ex_port](#)

```
typedef ubyte hsk\_ex\_port
```

Typedef for external interrupt ports.

16.9.3 Function Documentation

16.9.3.1 [hsk_ex_channel_disable\(\)](#)

```
void hsk\_ex\_channel\_disable (  
    const hsk\_ex\_channel channel )
```

Disables an external interrupt channel.

Parameters

<i>channel</i>	The channel to disable, one of External Interrupt Channels
----------------	--

16.9.3.2 hsk_ex_channel_enable()

```
void hsk_ex_channel_enable (
    const hsk_ex_channel channel,
    const ubyte edge,
    const void(*) (void) callback )
```

Enable an external interrupt channel.

It is good practice to enable a port for the channel first, because port changes on an active interrupt may cause an undesired interrupt.

The callback function can be set to 0 if a change of the function is not desired. For channels EXINT0 and EXINT1 the callback is ignored, implement interrupts 0 and 2 instead.

Parameters

<i>channel</i>	The channel to activate, one of External Interrupt Channels
<i>edge</i>	The triggering edge, one of External Interrupt Triggers
<i>callback</i>	The callback function for an interrupt event

Setting up EXINT0/1 is somewhat confusing. Refer to UM 1.1 section 5.6.2 to make sense of this.

16.9.3.3 hsk_ex_port_close()

```
void hsk_ex_port_close (
    const hsk_ex_port port )
```

Disconnects an input port from an external interrupt.

Parameters

<i>port</i>	The port to close, one of External Interrupt Input Ports
-------------	--

16.9.3.4 hsk_ex_port_open()

```
void hsk_ex_port_open (
    const hsk_ex_port port )
```

Opens an input port for an external interrupt.

Parameters

<i>port</i>	The port to open, one of External Interrupt Input Ports
-------------	---

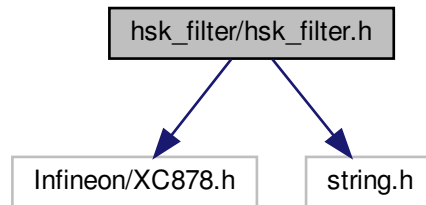
16.10 hsk_filter/hsk_filter.h File Reference

HSK Filter generator.

```
#include <Infineon/XC878.h>
```

```
#include <string.h>
```

Include dependency graph for hsk_filter.h:



Macros

- `#define FILTER_FACTORY(prefix, valueType, sumType, sizeType, size)`
Generates a filter.
- `#define FILTER_GROUP_FACTORY(prefix, filters, valueType, sumType, sizeType, size)`
Generates a group of filters.

16.10.1 Detailed Description

HSK Filter generator.

This file offers preprocessor macros to filter analogue values, by calculating the average of a set of a given length.

The buffer for the filter is stored in xdata memory.

Author

kami

16.10.2 Macro Definition Documentation

16.10.2.1 FILTER_FACTORY

```
#define FILTER_FACTORY(
    prefix,
    valueType,
    sumType,
    sizeType,
    size )
```

Generates a filter.

The filter can be accessed with:

- void <prefix>_init(void)
 - Initializes the filter with 0
- <valueType> <prefix>_update(const <valueType> value)
 - Update the filter and return the current average

Parameters

<i>prefix</i>	A prefix for the generated internals and functions
<i>valueType</i>	The data type of the stored values
<i>sumType</i>	A data type that can contain the sum of all buffered values
<i>sizeType</i>	A data type that can hold the length of the buffer
<i>size</i>	The length of the buffer

16.10.2.2 FILTER_GROUP_FACTORY

```
#define FILTER_GROUP_FACTORY(
    prefix,
    filters,
    valueType,
    sumType,
    sizeType,
    size )
```

Generates a group of filters.

The filters can be accessed with:

- void <prefix>_init(void)
 - Initializes all filters with 0
- <valueType> <prefix>_update(const ubyte filter, const <valueType> value)
 - Update the given filter and return the current average

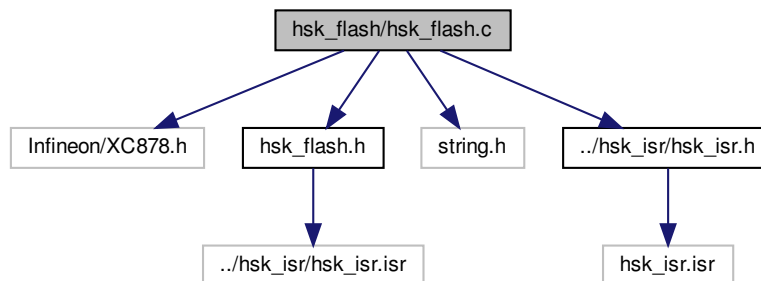
Parameters

<i>prefix</i>	A prefix for the generated internals and functions
<i>filters</i>	The number of filters
<i>valueType</i>	The data type of the stored values
<i>sumType</i>	A data type that can contain the sum of all buffered values
<i>sizeType</i>	A data type that can hold the length of the buffer
<i>size</i>	The length of the buffer

16.11 hsk_flash/hsk_flash.c File Reference

HSK Flash Facility implementation.

```
#include <Infineon/XC878.h>
#include "hsk_flash.h"
#include <string.h>
#include "../hsk_isr/hsk_isr.h"
Include dependency graph for hsk_flash.c:
```



Macros

- **#define MOVCI** .db 0xA5
MOVC @(DPTR++),A instruction.
- **#define DPL** dpl
DPTR low byte.
- **#define DPH** dph
DPTR high byte.
- **#define VAR_AT**(type, name, addr) type __at(addr) name
Create variable at a certain address, SDCC version.
- **#define VAR_ASM**(name) _##name
Insert global variable address into inline assembler SDCC style.
- **#define PAGE_RAM** 2
XC878-16FF code page that has the Boot ROM and XRAM mapped into it.
- **#define PAGE_FLASH** 0
XC878-16FF code page that has the flash.
- **#define ADDR_PFLASH** 0x0000
XC878-16FF start address of the P-Flash.
- **#define LEN_PFLASH** (60u << 10)
XC878-16FF length of the P-Flash.
- **#define BYTES_PAGE_PFLASH** (1 << 9)
XC878-16FF the number of bytes in a P-Flash page.
- **#define BYTES_WORDLINE_PFLASH** (1 << 6)
XC878-16FF the number of bytes in a P-Flash wordline.
- **#define ADDR_DFLASH** 0xF000
XC878-16FF start address of the D-Flash.
- **#define LEN_DFLASH** (4u << 10)
XC878-16FF length of the D-Flash.
- **#define BYTES_PAGE_DFLASH** (1 << 6)
XC878-16FF the number of bytes in a D-Flash page.
- **#define BYTES_WORDLINE_DFLASH** (1 << 5)

- XC878-16FF Id() of the number of bytes in a D-Flash wordline.*

 - #define `ADDR_ROM` 0xC000

XC878-16FF start address of the Boot ROM.
- #define `LEN_ROM` (8u << 10)

XC878-16FF length of the Boot ROM.
- #define `ADDR_XRAM` 0xF000

XC878-16FF start address of the XRAM.
- #define `LEN_XRAM` (3u << 10)

XC878-16FF length of the XRAM.
- #define `BIT_PROG` 0

FCON/EECON Program Bit.
- #define `BIT_ERASE` 1

FCON/EECON Erase Bit.
- #define `BIT_MAS1` 2

FCON/EECON Mass Erase Bit.
- #define `BIT_NVSTR` 3

FCON/EECON Non-Volatile Store Bit.
- #define `BIT_YE` 5

FCON/EECON Y-Address Enable Bit.
- #define `BIT_EEBSY` 6

EECON D-Flash Busy Bit.
- #define `BIT_FTEN` 5

FCS Flash Timer Enable Bit.
- #define `BIT_EEABORT` 0

FCS1 D-Flash Program/Erase Abort bit.
- #define `BIT_OFVAL` 0

FTVAL Overflow Value bits.
- #define `CNT_OFVAL` 7

OFVAL bit count.
- #define `BIT_MODE` 7

FTVAL MODE bit.
- #define `BIT_NMIFLASH` 2

NMICON Flash Timer NMI Enable bit.
- #define `STATE_IDLE` 0

The state to use when nothing is to be done.
- #define `STATE_REQUEST` 1

The state to use to kick off a write.
- #define `STATE_DETECT` 10

The state that decides whether a delete or idle is appropriate.
- #define `STATE_WRITE` 20

The state to use when starting to write to the D-Flash.
- #define `STATE_DELETE` 40

The state to use when erasing D-Flash pages.
- #define `STATE_RESET` 60

The state to use when mass erasing the D-Flash.
- #define `FREE_LATEST` 0

The block indicated `hsk_flash::latest` is available for writing.
- #define `FREE_BEHIND` 1

The block behind the block indicated by `hsk_flash::latest` is available for writing.
- #define `FREE_NONE` 2

There is no block available for writing.

Functions

- void `hsk_flash_isr_nmiflash` (void)
Flash delete/write state machine.
- ubyte `hsk_flash_init` (void *const `ptr`, const uword `size`, const ubyte version)
Recovers a struct from a previous session and sets everything up for storage of changes.
- bool `hsk_flash_write` (void)
Writes the current data to the D-Flash.

Variables

- static const ubyte `dflash` [(4u << 10)]
Bytewise access to the D-Flash area.
- SFR `FCON` = 0xD1
P-Flash Control Register.
- SFR `EECON` = 0xD2
D-Flash Control Register.
- SFR `FCS` = 0xD3
Flash Control and Status Register.
- SFR `FEAL` = 0xD4
Flash Error Address Register Low.
- SFR `FEAH` = 0xD5
Flash Error Address Register High.
- SFR16 `FEALH` = 0xD4
Flash Error Address Register Low and High (16 bits).
- SFR `FTVAL` = 0xD6
Flash Timer Value Register.
- SFR `FCS1` = 0xDD
Flash Control and Status Register 1.
- struct {
 ubyte * `ptr`
 The pointer to the data structure to persist.
 uword `size`
 The size of the data structure to persist.
 uword `wrap`
 The useable amount of D-Flash.
 uword `oldest`
 The offset of the oldest data in the D-Flash.
 uword `latest`
 The offset of the latest data in the D-Flash.
 ubyte `free`
 This byte indicates where free space can be found in the D-Flash.
 ubyte `ident`
 The prefix/postfix to identify the data structure in the flash.
 ubyte `state`
 The current state of the flash ISR state machine.
} `flash`

Holds the persistence configuration.
- static volatile ubyte * `flashDptr`
A pointer to the flash target address.
- static volatile ubyte * `xdataDptr`
A pointer to the xdata src address.

16.11.1 Detailed Description

HSK Flash Facility implementation.

This file implements the flash management functions defined in [hsk_flash.h](#).

Author

kami

16.11.2 Flash Registers

All registers are in the mapped register are, i.e. RMAP=1 must be set to access them.

16.11.3 Flash Timer

Non-blocking flash reading/writing is controlled by a dedicated flash timer. Timings, especially flash writing, are so critical that all the flash delete/write procedures are implemented in a single state machine within [hsk_flash_isr](#) ↔ [nmiflash\(\)](#), which is called by a non-maskable interrupt upon timer overflow.

16.11.4 DPTR Byte Order

Due to the [Byte Order](#) differences between SDCC and C51, the [DPL](#) and [DPH](#) macros are used to adjust DPTR assignments in inline assembler.

16.11.5 Macro Definition Documentation

16.11.5.1 ADDR_DFLASH

```
#define ADDR_DFLASH 0xF000
```

XC878-16FF start address of the D-Flash.

16.11.5.2 ADDR_PFLASH

```
#define ADDR_PFLASH 0x0000
```

XC878-16FF start address of the P-Flash.

16.11.5.3 ADDR_ROM

```
#define ADDR_ROM 0xC000
```

XC878-16FF start address of the Boot ROM.

16.11.5.4 ADDR_XRAM

```
#define ADDR_XRAM 0xF000
```

XC878-16FF start address of the XRAM.

16.11.5.5 BIT_EEABORT

```
#define BIT_EEABORT 0
```

FCS1 D-Flash Program/Erase Abort bit.

16.11.5.6 BIT_EEBSY

```
#define BIT_EEBSY 6
```

EECON D-Flash Busy Bit.

16.11.5.7 BIT_ERASE

```
#define BIT_ERASE 1
```

FCON/EECON Erase Bit.

16.11.5.8 BIT_FTEN

```
#define BIT_FTEN 5
```

FCS Flash Timer Enable Bit.

16.11.5.9 BIT_MAS1

```
#define BIT_MAS1 2
```

FCON/EECON Mass Erase Bit.

16.11.5.10 BIT_MODE

```
#define BIT_MODE 7
```

FTVAL MODE bit.

Controls the flash timer speed.

Mode	Value	Effect
Program	0	1 count per <i>CCLK</i> (24MHz) clock cycle
Erase	1	1 count per $CCLK/2^{12}$ clock cycles

16.11.5.11 BIT_NMIFLASH

```
#define BIT_NMIFLASH 2
```

NMICON Flash Timer NMI Enable bit.

16.11.5.12 BIT_NVSTR

```
#define BIT_NVSTR 3
```

FCON/EECON Non-Volatile Store Bit.

16.11.5.13 BIT_OFVAL

```
#define BIT_OFVAL 0
```

FTVAL Overflow Value bits.

16.11.5.14 BIT_PROG

```
#define BIT_PROG 0
```

FCON/EECON Program Bit.

16.11.5.15 BIT_YE

```
#define BIT_YE 5
```

FCON/EECON Y-Address Enable Bit.

16.11.5.16 BYTES_PAGE_DFLASH

```
#define BYTES_PAGE_DFLASH (1 << 6)
```

XC878-16FF the number of bytes in a D-Flash page.

16.11.5.17 BYTES_PAGE_PFLASH

```
#define BYTES_PAGE_PFLASH (1 << 9)
```

XC878-16FF the number of bytes in a P-Flash page.

16.11.5.18 BYTES_WORDLINE_DFLASH

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

XC878-16FF Id() of the number of bytes in a D-Flash wordline.

16.11.5.19 BYTES_WORDLINE_PFLASH

```
#define BYTES_WORDLINE_PFLASH (1 << 6)
```

XC878-16FF the number of bytes in a P-Flash wordline.

16.11.5.20 CNT_OFVAL

```
#define CNT_OFVAL 7
```

OFVAL bit count.

16.11.5.21 DPH

```
#define DPH dph
```

DPTR high byte.

See also

[DPTR Byte Order](#)

16.11.5.22 DPL

```
#define DPL dpl
```

DPTR low byte.

See also

[DPTR Byte Order](#)

16.11.5.23 FREE_BEHIND

```
#define FREE_BEHIND 1
```

The block behind the block indicated by [hsk_flash::latest](#) is available for writing.

16.11.5.24 FREE_LATEST

```
#define FREE_LATEST 0
```

The block indicated [hsk_flash::latest](#) is available for writing.

16.11.5.25 FREE_NONE

```
#define FREE_NONE 2
```

There is no block available for writing.

16.11.5.26 LEN_DFLASH

```
#define LEN_DFLASH (4u << 10)
```

XC878-16FF length of the D-Flash.

16.11.5.27 LEN_PFLASH

```
#define LEN_PFLASH (60u << 10)
```

XC878-16FF length of the P-Flash.

16.11.5.28 LEN_ROM

```
#define LEN_ROM (8u << 10)
```

XC878-16FF length of the Boot ROM.

16.11.5.29 LEN_XRAM

```
#define LEN_XRAM (3u << 10)
```

XC878-16FF length of the XRAM.

16.11.5.30 MOVCI

```
#define MOVCI .db 0xA5
```

MOVC @(DPTR++),A instruction.

16.11.5.31 PAGE_FLASH

```
#define PAGE_FLASH 0
```

XC878-16FF code page that has the flash.

16.11.5.32 PAGE_RAM

```
#define PAGE_RAM 2
```

XC878-16FF code page that has the Boot ROM and XRAM mapped into it.

16.11.5.33 STATE_DELETE

```
#define STATE_DELETE 40
```

The state to use when erasing D-Flash pages.

16.11.5.34 STATE_DETECT

```
#define STATE_DETECT 10
```

The state that decides whether a delete or idle is appropriate.

16.11.5.35 STATE_IDLE

```
#define STATE_IDLE 0
```

The state to use when nothing is to be done.

16.11.5.36 STATE_REQUEST

```
#define STATE_REQUEST 1
```

The state to use to kick off a write.

16.11.5.37 STATE_RESET

```
#define STATE_RESET 60
```

The state to use when mass erasing the D-Flash.

16.11.5.38 STATE_WRITE

```
#define STATE_WRITE 20
```

The state to use when starting to write to the D-Flash.

16.11.5.39 VAR_ASM

```
#define VAR_ASM(  
    name ) _##name
```

Insert global variable address into inline assembler SDCC style.

16.11.5.40 VAR_AT

```
#define VAR_AT(  
    type,  
    name,  
    addr ) type __at(addr) name
```

Create variable at a certain address, SDCC version.

16.11.6 Function Documentation

16.11.6.1 hsk_flash_init()

```
ubyte hsk_flash_init (  
    void *const ptr,  
    const uword size,  
    const ubyte version )
```

Recovers a struct from a previous session and sets everything up for storage of changes.

There are two modes of recovery. After a fresh boot the data can be recovered from flash, if previously stored there. After a simple reset the data can still be found in XRAM and recovery can be sped up.

If recovery fails entirely all members of the struct will be set to 0.

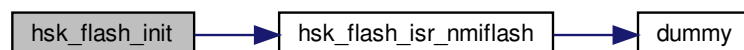
Parameters

<i>version</i>	Version number of the persisted data structure, used to prevent initialization with incompatible data
<i>ptr</i>	A pointer to the xdata struct/array to persist
<i>size</i>	The size of the data structure to persist

Return values

<i>FLASH_PWR_FIRST</i>	No valid data was recovered
<i>FLASH_PWR_RESET</i>	Continue operation after a reset
<i>FLASH_PWR_ON</i>	Data restore from the D-Flash succeeded

Here is the call graph for this function:



16.11.6.2 hsk_flash_isr_nmiflash()

```
void hsk_flash_isr_nmiflash (  
    void ) [private]
```

Flash delete/write state machine.

Every named state is the root of a state machine that performs a specific task.

See also

Section 4.4 *Flash Memory - Operating Modes* from the XC8787 reference manual: [XC878_um_v1_1.pdf](#)

- [STATE_IDLE](#) is a sleeping state that turns off the state machine. This state is a dead end, the state machine has to be reactivated externally to resume operation.
- [STATE_REQUEST](#) implements the procedure called "Abort Operation" from the XC878 UM 1.1. After completing the abort [STATE_WRITE](#) is entered.
- [STATE_DETECT](#) checks whether there is a page that should be deleted. It either goes into [STATE_DELETE](#) or [STATE_IDLE](#).
- [STATE_WRITE](#) implements the procedure called "Program Operation" from the XC878 UM 1.1. The next address to write is expected in [flashDptr](#). The next address to read from XRAM is expected in [xdataDptr](#).
- [STATE_DELETE](#) implements the procedure called "Erase Operation" from the XC878 UM 1.1.
- [STATE_RESET](#) implements the procedure called "Mass Erase Operation" from the XC878 UM 1.1.

Here is the call graph for this function:



16.11.6.3 hsk_flash_write()

```
bool hsk_flash_write (  
    void )
```

Writes the current data to the D-Flash.

Ongoing writes are interrupted. Ongoing deletes are interrupted unless there is insufficient space left to write the data.

Return values

1	The D-Flash write is on the way
0	Not enough free D-Flash space to write, try again later

16.11.7 Variable Documentation

16.11.7.1 dflash

```
const ubyte dflash[(4u<< 10)] [static]
```

Bytewise access to the D-Flash area.

16.11.7.2 EECON

```
SFR EECON = 0xD2
```

D-Flash Control Register.

16.11.7.3 FCON

```
SFR FCON = 0xD1
```

P-Flash Control Register.

16.11.7.4 FCS

```
SFR FCS = 0xD3
```

Flash Control and Status Register.

16.11.7.5 FCS1

```
SFR FCS1 = 0xDD
```

Flash Control and Status Register 1.

16.11.7.6 FEAH

```
SFR FEAH = 0xD5
```

Flash Error Address Register High.

16.11.7.7 FEAL

`SFR FEAL = 0xD4`

Flash Error Address Register Low.

16.11.7.8 FEALH

`SFR16 FEALH = 0xD4`

Flash Error Address Register Low and High (16 bits).

16.11.7.9 flash

`flash [static]`

Holds the persistence configuration.

16.11.7.10 flashDptr

`volatile ubyte* flashDptr [static]`

A pointer to the flash target address.

Not in a struct for easier inline assembler access.

16.11.7.11 free

`ubyte free`

This byte indicates where free space can be found in the D-Flash.

Available values are:

- [FREE_LATEST](#)
- [FREE_BEHIND](#)
- [FREE_NONE](#)

16.11.7.12 FTVAL

`SFR FTVAL = 0xD6`

Flash Timer Value Register.

16.11.7.13 ident

`ubyte ident`

The prefix/postfix to identify the data structure in the flash.

It consist of the last 6 bits of the version and two alternating bits to make sure the value can neither become 0x00 nor 0xff.

Pre-/postfixing the ident ensures that the data was completely written.

16.11.7.14 latest

`uword latest`

The offset of the latest data in the D-Flash.

16.11.7.15 oldest

`uword oldest`

The offset of the oldest data in the D-Flash.

16.11.7.16 ptr

`ubyte* ptr`

The pointer to the data structure to persist.

16.11.7.17 size

`uword size`

The size of the data structure to persist.

16.11.7.18 state

`ubyte state`

The current state of the flash ISR state machine.

16.11.7.19 wrap

`uword wrap`

The useable amount of D-Flash.

16.11.7.20 xdataDptr

```
volatile ubyte* xdataDptr [static]
```

A pointer to the xdata src address.

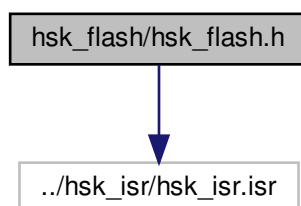
Not in a struct for easier inline assembler access.

16.12 hsk_flash/hsk_flash.h File Reference

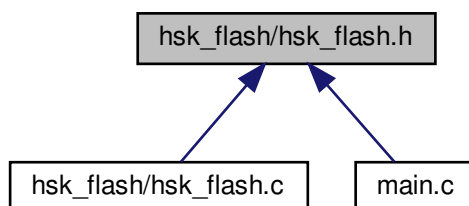
HSK Flash Facility headers.

```
#include "../hsk_isr/hsk_isr.isr"
```

Include dependency graph for hsk_flash.h:



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



Macros

- `#define XC878_16FF`
Ensure that a flash memory layout is defined.
- `#define FLASH_STRUCT_FACTORY(members)`
Used to create a struct that can be used with the `hsk_flash_init()` function.
- `#define FLASH_PWR_FIRST 0`
Returned by `hsk_flash_init()` when the μ C boots for the first time.
- `#define FLASH_PWR_RESET 1`
Returned by `hsk_flash_init()` after booting from a reset without power loss.
- `#define FLASH_PWR_ON 2`
Returned by `hsk_flash_init()` during power on, if valid data was recovered from the D-Flash.

Functions

- `ubyte hsk_flash_init (void *const ptr, const uword size, const ubyte version)`
Recovers a struct from a previous session and sets everything up for storage of changes.
- `bool hsk_flash_write (void)`
Writes the current data to the D-Flash.

16.12.1 Detailed Description

HSK Flash Facility headers.

This file contains function prototypes to manage information that survives a reset and allow storage within the D-Flash.

It provides the `FLASH_STRUCT_FACTORY` to create a struct with data that can be stored with `hsk_flash_write()` and recovered with `hsk_flash_init()`.

The D-Flash is used as a ring buffer, this distributes writes over the entire flash to gain the maximum achievable lifetime. The lifetime expectancy depends on your usage scenario and the size of the struct.

Refer to section 3.3 table 20 and table 21 of the [XC87x data sheet](#) for D-Flash life times.

Complete coverage of the D-Flash counts as a single D-Flash cycle. Thus the formula for the expected number of write calls is:

$$writes = \lfloor 4096 / sizeof(struct) \rfloor * expectedcycles$$

- `expectedcycles`
 - The expected number of possible write cycles depending on the usage scenario in table 20
- `sizeof(struct)`
 - The number of bytes the struct covers
- `floor()`
 - Round down to the next smaller integer

E.g. to store 20 bytes of configuration data, the struct factory adds 2 bytes overhead to be able to check the consistency of written data, so $sizeof(struct) = 22$. Expecting that most of the μ C use is within the first year, table 20 suggests that $expectedcycles = 100000$. In that case the expected number of possible `hsk_flash_write()` calls is 18.6 million.

Author

kami

16.12.2 Byte Order

C51 stores multiple byte variables in big endian order, whereas the DPTR register, several SFRs and SDCC use little endian.

If the data struct contains multibyte members such as int/word or long/ulong, this can lead to data corruption, when switching compilers.

Both the checksum and identifier are single byte values and thus will still match after a compiler switch, causing multibyte values to be restored from the flash with the wrong byte order.

A byte order change can be detected with a byte order word in the struct. A BOW initialized with 0x1234 would read 0x3412 after a an order change.

The suggested solution is to only create struct members with byte wise access. E.g. a ulong member may be defined in the following way:

```
ubyte ulongMember[sizeof(ulong)];
```

The value can be set like this:

```
myStruct.ulongMember[0] = ulongValue;
myStruct.ulongMember[1] = ulongValue >> 8;
myStruct.ulongMember[2] = ulongValue >> 16;
myStruct.ulongMember[3] = ulongValue >> 24;
```

Reading works similarly:

```
ulongValue = (ubyte)myStruct.ulongMember[0];
ulongValue |= (uword)myStruct.ulongMember[1] << 8;
ulongValue |= (ulong)myStruct.ulongMember[2] << 16;
ulongValue |= (ulong)myStruct.ulongMember[3] << 24;
```

Another alternative is to use a single ubyte[] array and store/read all data with the [hsk_can_data_setSignal\(\)/hsk←_can_data_getSignal\(\)](#) functions. Due to the bit addressing of CAN message data the maximum length of such an array would be 32 bytes (256bits).

An advantage would be that less memory is required, because data no longer needs to be byte aligned.

16.12.3 Macro Definition Documentation

16.12.3.1 FLASH_PWR_FIRST

```
#define FLASH_PWR_FIRST 0
```

Returned by [hsk_flash_init\(\)](#) when the µC boots for the first time.

This statements holds true *as far as can be told*. I.e. a first boot is diagnosed when all attempts to recover data have failed.

Two scenarios may cause this:

- No valid data has yet been written to the D-Flash
- The latest flash data is corrupted, may happen in case of power down during write

16.12.3.2 FLASH_PWR_ON

```
#define FLASH_PWR_ON 2
```

Returned by [hsk_flash_init\(\)](#) during power on, if valid data was recovered from the D-Flash.

A power on is detected when two criteria are met:

- Data could not be recovered from `xdata` memory
- Valid data was recovered from the D-Flash

16.12.3.3 FLASH_PWR_RESET

```
#define FLASH_PWR_RESET 1
```

Returned by [hsk_flash_init\(\)](#) after booting from a reset without power loss.

The typical mark of a reset is that `xdata` memory still holds data from the previous session. If such data is found it will just be picked up.

For performance reasons access to the struct is not guarded, which means that there can be no protection against data corruption, such as might be caused by a software bug like an overflow.

16.12.3.4 FLASH_STRUCT_FACTORY

```
#define FLASH_STRUCT_FACTORY(  
    members )
```

Value:

```
/**  
This struct is a template for data that can be written to the D-Flash.  
It is created by invoking the \ref FLASH_STRUCT_FACTORY macro.  
*/  
volatile struct hsk_flash_struct {  
    /**  
For data integrity/compatibility detection.  
@private  
        */  
        ubyte hsk_flash_prefix;  
        \br/>        members\  
        \  
        /**  
For data integrity detection.  
@private  
        */  
        ubyte hsk_flash_checksum;  
} xdata
```

Used to create a struct that can be used with the [hsk_flash_init\(\)](#) function.

The [hsk_flash_init\(\)](#) function expects certain fields to exist, in the struct, which are used to ensure the consistency of data in the flash.

The following example shows how to create a struct named `storableData`:

```
FLASH_STRUCT_FACTORY(  
    ubyte storableByte;  
    uword storableWord;  
) storableData;
```

Parameters

<i>members</i>	Struct member definitions
----------------	---------------------------

16.12.3.5 XC878_16FF

```
#define XC878_16FF
```

Ensure that a flash memory layout is defined.

Either XC878_16FF (64k flash) or XC878_13FF(52k flash) are supported. XC878_16FF is the default.

16.12.4 Function Documentation

16.12.4.1 hsk_flash_init()

```
ubyte hsk_flash_init (
    void *const ptr,
    const uword size,
    const ubyte version )
```

Recovers a struct from a previous session and sets everything up for storage of changes.

There are two modes of recovery. After a fresh boot the data can be recovered from flash, if previously stored there. After a simple reset the data can still be found in XRAM and recovery can be sped up.

If recovery fails entirely all members of the struct will be set to 0.

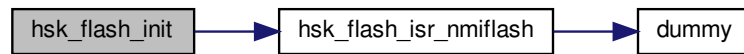
Parameters

<i>version</i>	Version number of the persisted data structure, used to prevent initialization with incompatible data
<i>ptr</i>	A pointer to the xdata struct/array to persist
<i>size</i>	The size of the data structure to persist

Return values

<i>FLASH_PWR_FIRST</i>	No valid data was recovered
<i>FLASH_PWR_RESET</i>	Continue operation after a reset
<i>FLASH_PWR_ON</i>	Data restore from the D-Flash succeeded

Here is the call graph for this function:



16.12.4.2 hsk_flash_write()

```
bool hsk_flash_write (
    void )
```

Writes the current data to the D-Flash.

Ongoing writes are interrupted. Ongoing deletes are interrupted unless there is insufficient space left to write the data.

Return values

1	The D-Flash write is on the way
0	Not enough free D-Flash space to write, try again later

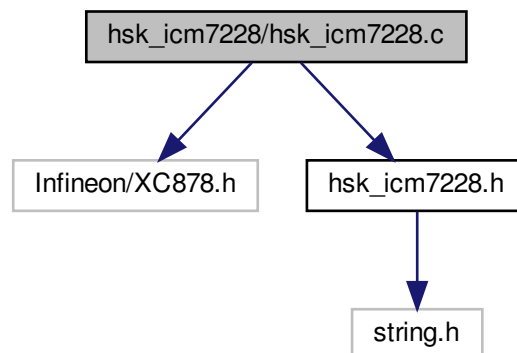
16.13 hsk_icm7228/hsk_icm7228.c File Reference

HSK ICM7228 8-Digit LED Display Decoder Driver implementation.

```
#include <Infineon/XC878.h>
```

```
#include "hsk_icm7228.h"
```

Include dependency graph for `hsk_icm7228.c`:



Macros

- `#define ILLUMINATE_OFFSET 16`
The offset for illuminating a number of segments.

Functions

- void `hsk_icm7228_writeString` (ubyte *const `buffer`, char const *`str`, ubyte `pos`, ubyte `len`)
Convert an ASCII string to 7 segment encoding and store it in an xdata buffer.
- void `hsk_icm7228_writeDec` (ubyte *const `buffer`, uword `value`, char `power`, ubyte const `pos`, ubyte `len`)
Write a 7 segment encoded, right aligned decimal number into an xdata buffer.
- void `hsk_icm7228_writeHex` (ubyte *const `buffer`, uword `value`, char `power`, ubyte const `pos`, ubyte `len`)
Write a 7 segment encoded, right aligned hexadecimal number into an xdata buffer.
- void `hsk_icm7228_illuminate` (ubyte *const `buffer`, ubyte `segments`, ubyte `pos`, ubyte `len`)
Illumante the given number of segments.

Variables

- static const ubyte `codepage` []
This is a codepage to translate 7bit ASCII characters into corresponding 7 segment display patterns.

16.13.1 Detailed Description

HSK ICM7228 8-Digit LED Display Decoder Driver implementation.

This file implements the static functions of the ICM7228 display decoder driver.

See also

Intersil ICM7228 Data Sheet: [ICM7228.pdf](#)

Author

kami

16.13.2 Macro Definition Documentation

16.13.2.1 ILLUMINATE_OFFSET

```
#define ILLUMINATE_OFFSET 16
```

The offset for illuminating a number of segments.

16.13.3 Function Documentation

16.13.3.1 hsk_icm7228_illuminate()

```
void hsk_icm7228_illuminate (
    ubyte *const buffer,
    ubyte segments,
    ubyte pos,
    ubyte len )
```

Illumante the given number of segments.

Parameters

<i>buffer</i>	The target buffer for the encoded string
<i>segments</i>	The number of segments to illuminate
<i>pos</i>	The target position in the buffer
<i>len</i>	The number of digits available to encode the number

16.13.3.2 hsk_icm7228_writeDec()

```
void hsk_icm7228_writeDec (
    ubyte *const buffer,
    uword value,
    char power,
    ubyte const pos,
    ubyte len )
```

Write a 7 segment encoded, right aligned decimal number into an xdata buffer.

The power parameter controls the placing of the '.' by 10 to the power. E.g. value = 12, power = -1 and len = 3 would result in the encoding of " 1.2". If power = 0, no dot is drawn. If the power is positive (typically 1), the resulting string would be filled with '0' characters. I.e. the previous example with power = 1 would result in an encoding of "012".

Parameters

<i>buffer</i>	The target buffer for the encoded string
<i>value</i>	The number to encode
<i>power</i>	The 10 base power of the number to encode
<i>pos</i>	The target position in the buffer
<i>len</i>	The number of digits available to encode the number

16.13.3.3 hsk_icm7228_writeHex()

```
void hsk_icm7228_writeHex (
    ubyte *const buffer,
    uword value,
    char power,
    ubyte const pos,
    ubyte len )
```

Write a 7 segment encoded, right aligned hexadecimal number into an xdata buffer.

The power parameter controls the placing of the '.' by 16 to the power. E.g. value = 0x1A, power = -1 and len = 3 would result in the encoding of " 1.A". If power = 0, no dot is drawn. If the power is positive (typically 1), the resulting string would be filled with '0' characters. I.e. the previous example with power = 1 would result in an encoding of "01A".

Parameters

<i>buffer</i>	The target buffer for the encoded string
<i>value</i>	The number to encode

Parameters

<i>power</i>	The 16 base power of the number to encode
<i>pos</i>	The target position in the buffer
<i>len</i>	The number of digits available to encode the number

16.13.3.4 hsk_icm7228_writeString()

```
void hsk_icm7228_writeString (
    ubyte *const buffer,
    char const * str,
    ubyte pos,
    ubyte len )
```

Convert an ASCII string to 7 segment encoding and store it in an xdata buffer.

This function is usually invoked through the <prefix>_writeString() function created by ICM7228_FACTORY.

The function will write into the buffer until it has been filled with *len* characters or it encounters a 0 character reading from *str*. If the character '.' is encountered it is merged with the previous character, unless that character is a '.' itself. Thus a single dot does not use additional buffer space. The 7 character string "foo ..." would result in 6 encoded bytes. Thus the proper *len* value for that string would be 6.

Parameters

<i>buffer</i>	The target buffer for the encoded string
<i>str</i>	The buffer to read the ASCII string from
<i>pos</i>	The position in the buffer to write the encoded string to
<i>len</i>	The target length of the encoded string

16.13.4 Variable Documentation

16.13.4.1 codepage

```
const ubyte codepage[] [static]
```

Initial value:

```
= {
    0xFB, 0xB0, 0xED, 0xF5, 0xB6, 0xD7, 0xDF, 0xF0,
    0xFF, 0xF7, 0xFE, 0x9F, 0xCB, 0xBD, 0xCF, 0xCE,
    0x80, 0xC0, 0xE0, 0xF0, 0xF1, 0xF9, 0xFB, 0xFF,
    0x7F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x80, 0x00, 0x00, 0x00, 0x00, 0x82, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0xFB, 0xB0, 0xED, 0xF5, 0xB6, 0xD7, 0xDF, 0xF0,
    0xFF, 0xF7, 0xFE, 0x9F, 0xCB, 0xBD, 0xCF, 0xCE,
    0x00, 0xFE, 0x9F, 0xCB, 0xBD, 0xCF, 0xCE, 0xDF,
    0xBE, 0x8A, 0xB1, 0xDE, 0x8B, 0xFA, 0xFA, 0x9D,
    0xEE, 0xF6, 0x8C, 0xD7, 0x8F, 0xBB, 0x99, 0xBB,
    0xB4, 0xB6, 0xC5, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0xFE, 0x9F, 0xCB, 0xBD, 0xCF, 0xCE, 0xDF,
    0xBE, 0x8A, 0xB1, 0xDE, 0x8B, 0xFA, 0xFA, 0x9D,
    0xEE, 0xF6, 0x8C, 0xD7, 0x8F, 0xBB, 0x99, 0xBB,
    0xB4, 0xB6, 0xC5, 0x00, 0x00, 0x00, 0x00, 0x00
}
```


This is a codepage to translate 7bit ASCII characters into corresponding 7 segment display patterns.

The ASCII character can be used to receive the desired 7 segment code. E.g. `codepage['A']` retrieves the letter 'A' from the table.

Some letters like 'X' are badly recognisable, others cannot be well represented at all. E.g. the letters 'M' and 'W' are identical to the letters 'N' and 'U'.

Capitals and small characters are identical. Characters without proper encoding are filled with 0x00, which leaves only the '.' of a 7 segment display active.

The 6 characters beyond 0-9 return "ABCDEF", which permits for easier display of HEX digits.

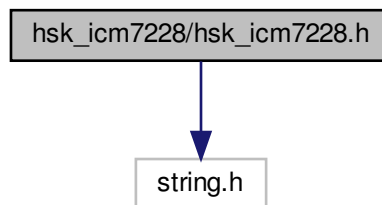
The first 16 characters from index 0 return the characters "0123456789ABCDEF" as well.

16.14 hsk_icm7228/hsk_icm7228.h File Reference

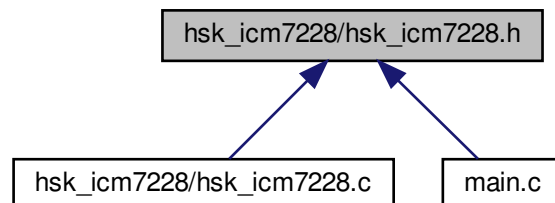
HSK ICM7228 8-Digit LED Display Decoder Driver generator.

```
#include <string.h>
```

Include dependency graph for hsk_icm7228.h:



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



Macros

- `#define ICM7228_FACTORY(prefix, regData, regMode, bitMode, regWrite, bitWrite)`
Generate an ICM7228 driver instance.

Functions

- void `hsk_icm7228_writeString` (ubyte *const `buffer`, char const *str, ubyte `pos`, ubyte len)
Convert an ASCII string to 7 segment encoding and store it in an xdata buffer.
- void `hsk_icm7228_writeDec` (ubyte *const `buffer`, uword value, char power, ubyte const `pos`, ubyte len)
Write a 7 segment encoded, right aligned decimal number into an xdata buffer.
- void `hsk_icm7228_writeHex` (ubyte *const `buffer`, uword value, char power, ubyte const `pos`, ubyte len)
Write a 7 segment encoded, right aligned hexadecimal number into an xdata buffer.
- void `hsk_icm7228_illuminate` (ubyte *const `buffer`, ubyte segments, ubyte `pos`, ubyte len)
Illuminate the given number of segments.

16.14.1 Detailed Description

HSK ICM7228 8-Digit LED Display Decoder Driver generator.

This file is a code generating facility, that offers preprocessor macros that produce code for the Intersil ICM7228 display decoder.

Generating code in this fashion avoids the hard coding of I/O registers and bits and even allows the use of multiple ICM7228 ICs.

See also

Intersil ICM7228 Data Sheet: [ICM7228.pdf](#)

Author

kami

16.14.2 Macro Definition Documentation

16.14.2.1 ICM7228_FACTORY

```
#define ICM7228_FACTORY(  
    prefix,  
    regData,  
    regMode,  
    bitMode,  
    regWrite,  
    bitWrite )
```

Generate an ICM7228 driver instance.

This creates functions to use a connect ICM7228 IC.

- void <prefix>_init(void)
 - Initialize the buffer and I/O register bits
- void <prefix>_refresh(void)
 - Commit buffered data to the 7 segment displays
- void <prefix>_writeString(char * str, ubyte pos, ubyte len)
 - Wrapper around [hsk_icm7228_writeString\(\)](#)
- void <prefix>_writeDec(uword value, char power, ubyte pos, ubyte len)
 - Wrapper around [hsk_icm7228_writeDec\(\)](#)
- void <prefix>_writeHex(uword value, char power, ubyte pos, ubyte len)
 - Wrapper around [hsk_icm7228_writeHex\(\)](#)

Parameters

<i>prefix</i>	A prefix for the names of generated functions
<i>regData</i>	The register that is connected to the data input
<i>regMode</i>	The register that is connected to the mode pin
<i>bitMode</i>	The bit of the regMode register that is connected to the mode pin
<i>regWrite</i>	The register that is connected to the write pin
<i>bitWrite</i>	The bit of the regWrite register that is connected to the write pin

16.14.3 Function Documentation

16.14.3.1 hsk_icm7228_illuminate()

```
void hsk_icm7228_illuminate (
    ubyte *const buffer,
    ubyte segments,
    ubyte pos,
    ubyte len )
```

Illumante the given number of segments.

Parameters

<i>buffer</i>	The target buffer for the encoded string
<i>segments</i>	The number of segments to illuminate
<i>pos</i>	The target position in the buffer
<i>len</i>	The number of digits available to encode the number

16.14.3.2 hsk_icm7228_writeDec()

```
void hsk_icm7228_writeDec (
    ubyte *const buffer,
```

```

    uword value,
    char power,
    ubyte const pos,
    ubyte len )

```

Write a 7 segment encoded, right aligned decimal number into an xdata buffer.

The power parameter controls the placing of the '.' by 10 to the power. E.g. value = 12, power = -1 and len = 3 would result in the encoding of " 1.2". If power = 0, no dot is drawn. If the power is positive (typically 1), the resulting string would be filled with '0' characters. I.e. the previous example with power = 1 would result in an encoding of "012".

Parameters

<i>buffer</i>	The target buffer for the encoded string
<i>value</i>	The number to encode
<i>power</i>	The 10 base power of the number to encode
<i>pos</i>	The target position in the buffer
<i>len</i>	The number of digits available to encode the number

16.14.3.3 hsk_icm7228_writeHex()

```

void hsk_icm7228_writeHex (
    ubyte *const buffer,
    uword value,
    char power,
    ubyte const pos,
    ubyte len )

```

Write a 7 segment encoded, right aligned hexadecimal number into an xdata buffer.

The power parameter controls the placing of the '.' by 16 to the power. E.g. value = 0x1A, power = -1 and len = 3 would result in the encoding of " 1.A". If power = 0, no dot is drawn. If the power is positive (typically 1), the resulting string would be filled with '0' characters. I.e. the previous example with power = 1 would result in an encoding of "01A".

Parameters

<i>buffer</i>	The target buffer for the encoded string
<i>value</i>	The number to encode
<i>power</i>	The 16 base power of the number to encode
<i>pos</i>	The target position in the buffer
<i>len</i>	The number of digits available to encode the number

16.14.3.4 hsk_icm7228_writeString()

```

void hsk_icm7228_writeString (
    ubyte *const buffer,
    char const * str,
    ubyte pos,
    ubyte len )

```

Convert an ASCII string to 7 segment encoding and store it in an xdata buffer.

This function is usually invoked through the <prefix>_writeString() function created by ICM7228_FACTORY.

The function will write into the buffer until it has been filled with len characters or it encounters a 0 character reading from str. If the character '.' is encountered it is merged with the previous character, unless that character is a '.' itself. Thus a single dot does not use additional buffer space. The 7 character string "foo ..." would result in 6 encoded bytes. Thus the proper len value for that string would be 6.

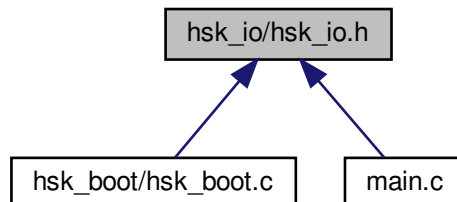
Parameters

<i>buffer</i>	The target buffer for the encoded string
<i>str</i>	The buffer to read the ASCII string from
<i>pos</i>	The position in the buffer to write the encoded string to
<i>len</i>	The target length of the encoded string

16.15 hsk_io/hsk_io.h File Reference

HSK I/O headers.

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



Macros

- `#define IO_PORT_IN_INIT(port, pins)`
Initializes a set of port pins as inputs.
- `#define IO_PORT_ON_GND 0`
Bit mask to set the logical 1 to GND level for all selected pins.
- `#define IO_PORT_ON_HIGH 0xff`
Bit mask to set the logical 1 to high level for all selected pins.
- `#define IO_PORT_GET(port, pins, on)`
Evaluates to a bit mask of logical pin states of a port.
- `#define IO_PORT_STRENGTH_WEAK 0`
Bit mask to set weak drive strength for all selected pins.
- `#define IO_PORT_STRENGTH_STRONG 0xff`

- Bit mask to set strong drive strength for all selected pins.*
 - `#define IO_PORT_DRAIN_DISABLE 0`
- Bit mask to disable drain mode for all selected pins.*
 - `#define IO_PORT_DRAIN_ENABLE 0xff`
- Bit mask to enable drain mode for all selected pins.*
 - `#define IO_PORT_OUT_INIT(port, pins, strength, drain, on, set)`
- Initializes a set of port pins as outputs.*
 - `#define IO_PORT_OUT_SET(port, pins, on, set)`
- Set a set of output port pins.*
 - `#define IO_PORT_PULL_DISABLE 0`
- Bit mask to disable pull up/down for all selected pins.*
 - `#define IO_PORT_PULL_ENABLE 0xff`
- Bit mask to enable pull up/down for all selected pins.*
 - `#define IO_PORT_PULL_DOWN 0`
- Bit mask to select pull down for all selected pins.*
 - `#define IO_PORT_PULL_UP 0xff`
- Bit mask to select pull up for all selected pins.*
 - `#define IO_PORT_PULL_INIT(port, pins, pull, dir)`
- Sets the pull-up/-down properties of port pins.*
 - `#define IO_VAR_SET(var, bits, on, set)`
- Set a set of variable bits.*
 - `#define IO_VAR_GET(var, bits, on)`
- Evaluates to a bit mask of logical states of a variable.*

16.15.1 Detailed Description

HSK I/O headers.

This file contains macro definitions to use and initialize I/O ports and variables bitwise.

All the macros take a port and a mask to select the affected pins. All operations are masked with the selected pins so it is safe to define 0xff (every bit 1) to activate a certain property.

Set and get macros take a bit field to define the value that represents the `on` or `true` state, so the logic code can always use a 1 for `true/on`.

The macros are grouped as:

- [Input Port Access](#)
- [Output Port Access](#)
- [Variable Access](#)

Author

kami

16.15.2 I/O Port Pull-Up/-Down Table

The device boots with all parallel ports configured as inputs. The following table lists the pins that come up with activated internal pull up:

Port\Bit	7	6	5	4	3	2	1	0
P0	1	1	x	x	x	1	x	x
P1	1	1	1	1	1	1	1	1
P3	x	1	x	x	x	x	x	x
P4	x	x	x	x	x	1	x	x
P5	1	1	1	1	1	1	1	1

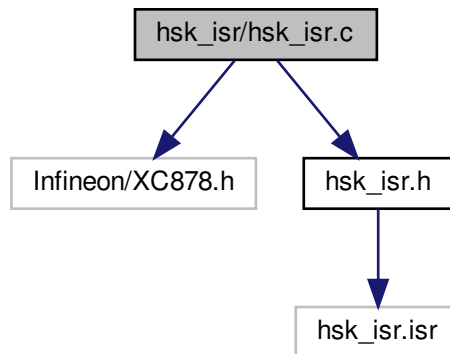
16.16 hsk_isr/hsk_isr.c File Reference

HSK Shared Interrupt Service Routine implementation.

```
#include <Infineon/XC878.h>
```

```
#include "hsk_isr.h"
```

Include dependency graph for hsk_isr.c:



Macros

- `#define BIT_RMAP 0`
SYSCON0 Special Function Register Map Control bit.
- `#define BIT_TF2 7`
T2_T2CON Timer 2 Overflow bit.
- `#define BIT_EXF2 6`
T2_T2CON T2EX bit.
- `#define BIT_CCTOVF 3`
T2CCU_CCTCON CCT Overflow bit.
- `#define BIT_NDOV 2`
FDCON Normal Divider Overflow bit.
- `#define BIT_EOFSYN 4`
FDCON End of Syn Byte bit.
- `#define BIT_ERRSYN 5`
FDCON Syn Byte Error bit.
- `#define BIT_CANSRC0 0`

- IRCON2 MultiCAN Node 0 bit.*
- #define [BIT_CANSRC1](#) 5
- IRCON1 Interrupt Flag 1 for MultiCAN bit.*
- #define [BIT_CANSRC2](#) 6
- IRCON1 Interrupt Flag 2 for MultiCAN bit.*
- #define [BIT_ADCSR0](#) 3
- IRCON1 Interrupt Flag 0 for ADC bit.*
- #define [BIT_ADCSR1](#) 4
- IRCON1 Interrupt Flag 1 for ADC bit.*
- #define [BIT_EXINT2](#) 2
- IRCON0 Interrupt Flag for External Interrupt 2 bit.*
- #define [BIT_RI](#) 0
- SCON Serial Interface Receiver Interrupt Flag.*
- #define [BIT_TI](#) 1
- SCON Serial Interface Transmitter Interrupt Flag.*
- #define [BIT_TF2](#) 7
- T2_ T2CON Timer 2 Overflow bit.*
- #define [BIT_EXF2](#) 6
- T2_ T2CON T2EX bit.*
- #define [BIT_NDOV](#) 2
- FDCON Normal Divider Overflow bit.*
- #define [BIT_EOC](#) 2
- CD_STATC End of Calculation Flag.*
- #define [BIT_IRDY](#) 0
- MDU_MDUSTAT Interrupt on Result Ready bit.*
- #define [BIT_IERR](#) 1
- MDU_MDUSTAT Interrupt on Error bit.*
- #define [BIT_EXINT3](#) 3
- IRCON0 Interrupt Flag for External Interrupt 3 or T2CC0 Capture/Compare Channel bit.*
- #define [BIT_EXINT4](#) 4
- IRCON0 Interrupt Flag for External Interrupt 4 or T2CC1 Capture/Compare Channel bit.*
- #define [BIT_EXINT5](#) 5
- IRCON0 Interrupt Flag for External Interrupt 5 or T2CC2 Capture/Compare Channel bit.*
- #define [BIT_EXINT6](#) 6
- IRCON0 Interrupt Flag for External Interrupt 6 or T2CC3 Capture/Compare Channel bit.*
- #define [BIT_CANSRC3](#) 4
- IRCON2 Interrupt Flag 3 for MultiCAN bit.*
- #define [BIT_NMIWDT](#) 0
- NMISR Watchdog Timer NMI Flag bit.*
- #define [BIT_NMIPLL](#) 1
- NMISR PLL NMI Flag bit.*
- #define [BIT_NMIFLASH](#) 2
- NMISR FLASH Timer NMI Flag bit.*
- #define [BIT_NMIVDDP](#) 5
- NMISR VDDP Prewarning NMI Flag bit.*
- #define [BIT_NMIECC](#) 6
- NMISR ECC NMI Flag bit.*

Functions

- void `hsk_isr_root1` (void)
This is a dummy function used for putting register bank 1 using ISRs into a common call tree for C51.
- void `dummy` (void)
This is a dummy function to point unused function pointers to.
- void `nmidummy` (void)
This is a dummy function to point unused function pointers to.
- void `ISR_hsk_isr5` (void)
Shared interrupt 5 routine.
- void `ISR_hsk_isr6` (void)
Shared interrupt 6 routine.
- void `ISR_hsk_isr8` (void)
Shared interrupt 8 routine.
- void `ISR_hsk_isr9` (void)
Shared interrupt 9 routine.
- void `ISR_hsk_isr14` (void)
Shared non-maskable interrupt routine.

Variables

- volatile struct `hsk_isr5_callback hsk_isr5` = {&`dummy`, &`dummy`, &`dummy`, &`dummy`, &`dummy`, &`dummy`, &`dummy`}
Define callback function pointers for ISR 5.
- volatile struct `hsk_isr6_callback hsk_isr6` = {&`dummy`, &`dummy`, &`dummy`, &`dummy`}
Define callback function pointers for ISR 6.
- volatile struct `hsk_isr8_callback hsk_isr8` = {&`dummy`, &`dummy`, &`dummy`, &`dummy`, &`dummy`, &`dummy`, &`dummy`, &`dummy`}
Define callback function pointers for ISR 8.
- volatile struct `hsk_isr9_callback hsk_isr9` = {&`dummy`, &`dummy`, &`dummy`, &`dummy`, &`dummy`}
Define callback function pointers for ISR 9.
- volatile struct `hsk_isr14_callback hsk_isr14` = {&`nmidummy`, &`nmidummy`, &`nmidummy`, &`nmidummy`, &`nmidummy`}
Define callback function pointers for NMI ISR.

16.16.1 Detailed Description

HSK Shared Interrupt Service Routine implementation.

This contains interrupts, shared between several interrupt sources. These interrupt sources can hook into the ISRs by storing a callback function in the `hsk_isr*` data structures.

Author

kami

16.16.2 ISR Callback Reaction Time

The following table describes what happens up to the point that the NMI ISR starts operation (based on SDCC code):

CCLK Cycles	Task	Instruction	Duration
0	Core: Poll interrupt request		2
2	Core: Call ISR	lcall	1 x 4
6	ISR setup: Push registers	6 x push	6 x 4
30	ISR setup: Reset PSW	1 x mov dir,#	1 x 4
34	ISR: Backup RMAP	...	3 x 2 + 4
44	ISR: Reset RMAP	...	2 x 2 + 4
52	ISR: Select callback

16.16.3 Macro Definition Documentation

16.16.3.1 BIT_ADCSR0

```
#define BIT_ADCSR0 3
```

IRCON1 Interrupt Flag 0 for ADC bit.

16.16.3.2 BIT_ADCSR1

```
#define BIT_ADCSR1 4
```

IRCON1 Interrupt Flag 1 for ADC bit.

16.16.3.3 BIT_CANSRC0

```
#define BIT_CANSRC0 0
```

IRCON2 MultiCAN Node 0 bit.

16.16.3.4 BIT_CANSRC1

```
#define BIT_CANSRC1 5
```

IRCON1 Interrupt Flag 1 for MultiCAN bit.

16.16.3.5 BIT_CANSRC2

```
#define BIT_CANSRC2 6
```

IRCON1 Interrupt Flag 2 for MultiCAN bit.

16.16.3.6 BIT_CANSRC3

```
#define BIT_CANSRC3 4
```

IRCON2 Interrupt Flag 3 for MultiCAN bit.

16.16.3.7 BIT_CCTOVF

```
#define BIT_CCTOVF 3
```

T2CCU_CCTCON CCT Overflow bit.

16.16.3.8 BIT_EOC

```
#define BIT_EOC 2
```

CD_STATC End of Calculation Flag.

16.16.3.9 BIT_EOFSYN

```
#define BIT_EOFSYN 4
```

FDCON End of Syn Byte bit.

16.16.3.10 BIT_ERRSYN

```
#define BIT_ERRSYN 5
```

FDCON Syn Byte Error bit.

16.16.3.11 BIT_EXF2 [1/2]

```
#define BIT_EXF2 6
```

T2_T2CON T2EX bit.

T2_T2CON Timer 2 External Flag.

16.16.3.12 BIT_EXF2 [2/2]

```
#define BIT_EXF2 6
```

T2_T2CON T2EX bit.

T2_T2CON Timer 2 External Flag.

16.16.3.13 BIT_EXINT2

```
#define BIT_EXINT2 2
```

IRCON0 Interrupt Flag for External Interrupt 2 bit.

16.16.3.14 BIT_EXINT3

```
#define BIT_EXINT3 3
```

IRCON0 Interrupt Flag for External Interrupt 3 or T2CC0 Capture/Compare Channel bit.

16.16.3.15 BIT_EXINT4

```
#define BIT_EXINT4 4
```

IRCON0 Interrupt Flag for External Interrupt 4 or T2CC1 Capture/Compare Channel bit.

16.16.3.16 BIT_EXINT5

```
#define BIT_EXINT5 5
```

IRCON0 Interrupt Flag for External Interrupt 5 or T2CC2 Capture/Compare Channel bit.

16.16.3.17 BIT_EXINT6

```
#define BIT_EXINT6 6
```

IRCON0 Interrupt Flag for External Interrupt 6 or T2CC3 Capture/Compare Channel bit.

16.16.3.18 BIT_IERR

```
#define BIT_IERR 1
```

MDU_MDUSTAT Interrupt on Error bit.

16.16.3.19 BIT_IRDY

```
#define BIT_IRDY 0
```

MDU_MDUSTAT Interrupt on Result Ready bit.

16.16.3.20 BIT_NDOV [1/2]

```
#define BIT_NDOV 2
```

FDCON Normal Divider Overflow bit.

FDCON Overflow Flag in Normal Divider Mode.

16.16.3.21 BIT_NDOV [2/2]

```
#define BIT_NDOV 2
```

FDCON Normal Divider Overflow bit.

FDCON Overflow Flag in Normal Divider Mode.

16.16.3.22 BIT_NMIECC

```
#define BIT_NMIECC 6
```

NMISR ECC NMI Flag bit.

16.16.3.23 BIT_NMIFLASH

```
#define BIT_NMIFLASH 2
```

NMISR FLASH Timer NMI Flag bit.

16.16.3.24 BIT_NMIPLL

```
#define BIT_NMIPLL 1
```

NMISR PLL NMI Flag bit.

16.16.3.25 BIT_NMIVDDP

```
#define BIT_NMIVDDP 5
```

NMISR VDDP Prewarning NMI Flag bit.

16.16.3.26 BIT_NMIWDT

```
#define BIT_NMIWDT 0
```

NMISR Watchdog Timer NMI Flag bit.

16.16.3.27 BIT_RI

```
#define BIT_RI 0
```

SCON Serial Interface Receiver Interrupt Flag.

16.16.3.28 BIT_RMAP

```
#define BIT_RMAP 0
```

SYSCON0 Special Function Register Map Control bit.

16.16.3.29 BIT_TF2 [1/2]

```
#define BIT_TF2 7
```

T2_T2CON Timer 2 Overflow bit.

T2_T2CON Timer 2 Overflow/Underflow Flag.

16.16.3.30 BIT_TF2 [2/2]

```
#define BIT_TF2 7
```

T2_T2CON Timer 2 Overflow bit.

T2_T2CON Timer 2 Overflow/Underflow Flag.

16.16.3.31 BIT_TI

```
#define BIT_TI 1
```

SCON Serial Interface Transmitter Interrupt Flag.

16.16.4 Function Documentation

16.16.4.1 dummy()

```
void dummy (  
    void ) [private]
```

This is a dummy function to point unused function pointers to.

16.16.4.2 hsk_isr_root1()

```
void hsk_isr_root1 (  
    void )
```

This is a dummy function used for putting register bank 1 using ISRs into a common call tree for C51.

16.16.4.3 ISR_hsk_isr14()

```
void ISR_hsk_isr14 (  
    void ) [private]
```

Shared non-maskable interrupt routine.

This interrupt has the following sources:

- Watchdog Timer NMI (NMIWDT)
- PLL NMI (NMIPLL)
- Flash Timer NMI (NMIFLASH)
- VDDP Prewarning NMI (NMIVDDP)
- Flash ECC NMI (NMIECC)

16.16.4.4 ISR_hsk_isr5()

```
void ISR_hsk_isr5 (  
    void ) [private]
```

Shared interrupt 5 routine.

Activate the interrupt by setting ET2 = 1.

This interrupt has the following sources:

- Timer 2 Overflow (TF2)
- Timer 2 External Event (EXF2)
- T2CCU CCT Overflow (CCTOVF)
- Normal Divider Overflow (NDOV)
- End of Syn Byte (EOFSYN)
- Syn Byte Error (ERRSYN)
- CAN Interrupt 0 (CANSRC0)

16.16.4.5 ISR_hsk_isr6()

```
void ISR_hsk_isr6 (  
    void ) [private]
```

Shared interrupt 6 routine.

Activate the interrupt by setting EADC = 1.

This interrupt has the following sources:

- CANSRC1
- CANSRC2
- ADCSR0
- ADCSR1

16.16.4.6 ISR_hsk_isr8()

```
void ISR_hsk_isr8 (  
    void ) [private]
```

Shared interrupt 8 routine.

Activate the interrupt by setting EX2 = 1.

This interrupt has the following sources:

- External Interrupt 2 (EXINT2)
- UART1 (RI)
- UART1 (TI)
- Timer 21 Overflow (TF2)
- T21EX (EXF2)
- UART1 Fractional Divider (Normal Divider Overflow) (NDOV)
- CORDIC (EOC)
- MDU Result Ready (IRDY)
- MDU Error (IERR)

16.16.4.7 ISR_hsk_isr9()

```
void ISR_hsk_isr9 (  
    void ) [private]
```

Shared interrupt 9 routine.

Activate the interrupt by setting EXM = 1.

This interrupt has the following sources:

- EXINT3/T2CC0
- EXINT4/T2CC1
- EXINT5/T2CC2
- EXINT6/T2CC3
- CANSRC3

16.16.4.8 nmidummy()

```
void nmidummy (  
    void ) [private]
```

This is a dummy function to point unused function pointers to.

16.16.5 Variable Documentation

16.16.5.1 hsk_isr14

```
volatile struct hsk_isr14_callback hsk_isr14 = {&nmidummy, &nmidummy, &nmidummy, &nmidummy,  
&nmidummy}
```

Define callback function pointers for NMI ISR.

Introduce callback function pointers for NMI ISR.

16.16.5.2 hsk_isr5

```
volatile struct hsk_isr5_callback hsk_isr5 = {&dummy, &dummy, &dummy, &dummy, &dummy, &dummy,  
&dummy}
```

Define callback function pointers for ISR 5.

Introduce callback function pointers for ISR 5.

16.16.5.3 hsk_isr6

```
volatile struct hsk_isr6_callback hsk_isr6 = {&dummy, &dummy, &dummy, &dummy}
```

Define callback function pointers for ISR 6.

Introduce callback function pointers for ISR 6.

16.16.5.4 hsk_isr8

```
volatile struct hsk_isr8_callback hsk_isr8 = {&dummy, &dummy, &dummy, &dummy, &dummy, &dummy,  
&dummy, &dummy, &dummy}
```

Define callback function pointers for ISR 8.

Introduce callback function pointers for ISR 8.

16.16.5.5 hsk_isr9

```
volatile struct hsk_isr9_callback hsk_isr9 = {&dummy, &dummy, &dummy, &dummy, &dummy}
```

Define callback function pointers for ISR 9.

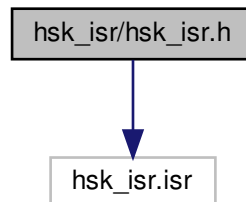
Introduce callback function pointers for ISR 9.

16.17 hsk_isr/hsk_isr.h File Reference

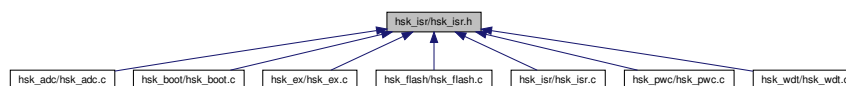
HSK Shared Interrupt Service Routine headers.

```
#include "hsk_isr_isr.h"
```

Include dependency graph for hsk_isr.h:



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



Data Structures

- struct [hsk_isr5_callback](#)
Shared interrupt 5 routine.
- struct [hsk_isr6_callback](#)
Shared interrupt 6 routine.
- struct [hsk_isr8_callback](#)
Shared interrupt 8 routine.
- struct [hsk_isr9_callback](#)
Shared interrupt 9 routine.
- struct [hsk_isr14_callback](#)
Shared non-maskable interrupt routine.

Variables

- volatile struct [hsk_isr5_callback](#) [hsk_isr5](#)
Introduce callback function pointers for ISR 5.
- volatile struct [hsk_isr6_callback](#) [hsk_isr6](#)
Introduce callback function pointers for ISR 6.
- volatile struct [hsk_isr8_callback](#) [hsk_isr8](#)
Introduce callback function pointers for ISR 8.
- volatile struct [hsk_isr9_callback](#) [hsk_isr9](#)
Introduce callback function pointers for ISR 9.
- volatile struct [hsk_isr14_callback](#) [hsk_isr14](#)
Introduce callback function pointers for NMI ISR.

16.17.1 Detailed Description

HSK Shared Interrupt Service Routine headers.

This header is used by other libraries to use interrupts with multiple sources. A callback function can be provided for each available interrupt source.

Author

kami

16.17.2 SFR Pages

An ISR callback function cannot make assumptions about current SFR pages like the regular functions that can expect all pages to be set to 0.

Instead a callback function needs to set all pages and restore whatever page was in use previously.

The following table lists the store and restore selectors by context and must be obeyed to avoid memory corruption:

Save	Restore	Context
SST0	RST0	ISRs
SST1	RST1	ISR callback functions
SST2	RST2	NMI ISR
SST3	RST3	NMI callback functions

Every callback function is called with RMAP = 0. If the callback function changes RMAP it does not have to take care of restoring it. RMAP is always restored to its original state by the shared ISRs.

16.17.3 Register Banks

Interrupts are each a root node of their own call tree. This is why they must preserve all the working registers.

the pushing and popping of the 8 `Rn` registers for each interrupt call costs 64 CCLK cycles.

To avoid this overhead different register banks are used. Call trees, i.e. interrupts, can use the same register bank if they cannot interrupt each other. Each used register bank costs 8 bytes of regular `data` memory. To minimize this cost all interrupts must have the same priority.

The following table is used:

Priority	Context	Bank
-	Regular code	0
0	ISR, callback	1
1	ISR, callback	-
2	ISR, callback	-
3	ISR, callback	-
NMI	NMI ISR, callback	2

Assigning higher priority to an ISR will affect (as in break) the operation of all lower priority ISRs.

16.17.4 Variable Documentation

16.17.4.1 hsk_isr14

```
volatile struct hsk_isr14_callback hsk_isr14
```

Introduce callback function pointers for NMI ISR.

Functions called back from the NMI ISR should use SST3/RST3 instead of SST1/RST1, because they might interrupt other ISRs.

Introduce callback function pointers for NMI ISR.

16.17.4.2 hsk_isr5

```
volatile struct hsk_isr5_callback hsk_isr5
```

Introduce callback function pointers for ISR 5.

Introduce callback function pointers for ISR 5.

16.17.4.3 hsk_isr6

```
volatile struct hsk_isr6_callback hsk_isr6
```

Introduce callback function pointers for ISR 6.

Introduce callback function pointers for ISR 6.

16.17.4.4 hsk_isr8

```
volatile struct hsk_isr8_callback hsk_isr8
```

Introduce callback function pointers for ISR 8.

Introduce callback function pointers for ISR 8.

16.17.4.5 hsk_isr9

```
volatile struct hsk_isr9_callback hsk_isr9
```

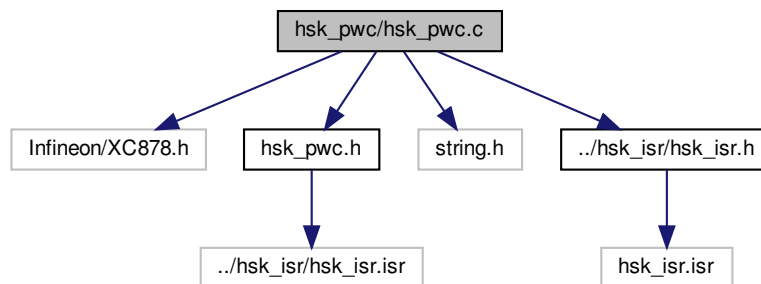
Introduce callback function pointers for ISR 9.

Introduce callback function pointers for ISR 9.

16.18 hsk_pwc/hsk_pwc.c File Reference

HSK Pulse Width Counter implementation.

```
#include <Infineon/XC878.h>
#include "hsk_pwc.h"
#include <string.h>
#include "../hsk_isr/hsk_isr.h"
Include dependency graph for hsk_pwc.c:
```



Macros

- `#define PWC_CHANNELS 4`
The number of available PWC channels.
- `#define CHAN_BUF_SIZE 8`
The size of a PWC ring buffer.
- `#define BIT_T2CCFG 4`
CR_MISC Timer 2 Capture/Compare Unit Clock Configuration bit.
- `#define BIT_CCTST 0`
T2CCU_CCTCON Capture/Compare Timer Start/Stop Control bit.
- `#define BIT_TIMSYN 1`
T2CCU_CCTCON Enable synchronized Timer Starts.
- `#define BIT_CCTOVEN 2`
T2CCU_CCTCON Capture/Compare Timer Overflow Interrupt Enable bit.
- `#define BIT_CCTOVF 3`
T2CCU_CCTCON Capture/Compare Timer Overflow Flag bit.
- `#define BIT_CCTPRE 4`
T2CCU_CCTCON T2CCU Capture/Compare Timer Control Register bits.
- `#define BIT_CCTBx 0`
T2CCU_CCTBSEL Channel x Time Base Select bit.
- `#define BIT_IMODE 4`
SYSCON0 Interrupt Structure 2 Mode Select bit.
- `#define BIT_CCM0 0`
T2CCU_CCEN Capture/Compare Enable bits start.
- `#define CNT_CCMx 2`
CCMx bit count.

- `#define EDGE_DEFAULT_MODE PWC_EDGE_BOTH`
Default to using both edges for pulse detection.
- `#define CNT_EXINTx 2`
EXICONn EXINTx mode bit count.
- `#define PWC_CC0_EXINT_REG EXICON0`
External Interrupt Control Register for setting the PWC_CC0 edge detection mode.
- `#define PWC_CC0_EXINT_BIT 6`
The edge detection mode bit position for PWC_CC0.
- `#define PWC_CC1_EXINT_REG EXICON1`
External Interrupt Control Register for setting the PWC_CC1 edge detection mode.
- `#define PWC_CC1_EXINT_BIT 0`
The edge detection mode bit position for PWC_CC1.
- `#define PWC_CC2_EXINT_REG EXICON1`
External Interrupt Control Register for setting the PWC_CC2 edge detection mode.
- `#define PWC_CC2_EXINT_BIT 2`
The edge detection mode bit position for PWC_CC2.
- `#define PWC_CC3_EXINT_REG EXICON1`
External Interrupt Control Register for setting the PWC_CC3 edge detection mode.
- `#define PWC_CC3_EXINT_BIT 4`
The edge detection mode bit position for PWC_CC3.
- `#define BIT_T2CCU_DIS 3`
PMCON1 T2CCU Disable Request bit.

Functions

- void `hsk_pwc_isr_ccn` (const `hsk_pwc_channel` channel, uword capture) using 1
This is the common implementation of the Capture ISRs.
- void `hsk_pwc_isr_cc0_p30` (void)
The ISR for Capture events on channel PWC_CC0_P30.
- void `hsk_pwc_isr_cc0_p40` (void)
The ISR for Capture events on channel PWC_CC0_P40.
- void `hsk_pwc_isr_cc0_p55` (void)
The ISR for Capture events on channel PWC_CC0_P55.
- void `hsk_pwc_isr_cc1_p32` (void)
The ISR for Capture events on channel PWC_CC1_P32.
- void `hsk_pwc_isr_cc1_p41` (void)
The ISR for Capture events on channel PWC_CC1_P41.
- void `hsk_pwc_isr_cc1_p56` (void)
The ISR for Capture events on channel PWC_CC1_P56.
- void `hsk_pwc_isr_cc2_p33` (void)
The ISR for Capture events on channel PWC_CC2_P33.
- void `hsk_pwc_isr_cc2_p44` (void)
The ISR for Capture events on channel PWC_CC2_P44.
- void `hsk_pwc_isr_cc2_p52` (void)
The ISR for Capture events on channel PWC_CC2_P52.
- void `hsk_pwc_isr_cc3_p34` (void)
The ISR for Capture events on channel PWC_CC3_P34.
- void `hsk_pwc_isr_cc3_p45` (void)
The ISR for Capture events on channel PWC_CC3_P45.
- void `hsk_pwc_isr_cc3_p57` (void)

- The ISR for Capture events on channel PWC_CC3_P57.*
- void [hsk_pwc_isr_cctOverflow](#) (void)
- The ISR for Capture/Compare overflow events.*
- void [hsk_pwc_ccn](#) (const [hsk_pwc_channel](#) channel, uword capture)
- This is the common implementation for soft capture events.*
- void [hsk_pwc_init](#) (ulong window)
- This function initializes the T2CCU Capture/Compare Unit for capture mode.*
- void [hsk_pwc_channel_open](#) (const [hsk_pwc_channel](#) channel, ubyte [averageOver](#))
- Configures a PWC channel without an input port.*
- void [hsk_pwc_port_open](#) (const [hsk_pwc_port](#) port, ubyte [averageOver](#))
- Opens an input port and the connected channel.*
- void [hsk_pwc_channel_close](#) (const [hsk_pwc_channel](#) channel)
- Close a PWC channel.*
- void [hsk_pwc_channel_edgeMode](#) (const [hsk_pwc_channel](#) channel, const ubyte edgeMode)
- Select the edge that is used to detect a pulse.*
- void [hsk_pwc_channel_captureMode](#) (const [hsk_pwc_channel](#) channel, const ubyte captureMode)
- Allows switching between external and soft trigger.*
- void [hsk_pwc_channel_trigger](#) (const [hsk_pwc_channel](#) channel)
- Triggers a channel in soft trigger mode.*
- void [hsk_pwc_enable](#) (void)
- Enables T2CCU module if disabled.*
- void [hsk_pwc_disable](#) (void)
- Turns off the T2CCU clock to preserve power.*
- ulong [hsk_pwc_channel_getValue](#) (const [hsk_pwc_channel](#) channel, const ubyte unit)
- Returns a measure of the values in a channel buffer.*

Variables

- static ubyte [prescaler](#)
- The prescaling factor.*
- static volatile ubyte [overflows](#)
- A CCT overflow counter.*
- struct {
- ulong [sum](#)
- The sum of the values stored in the ring buffer.*
- uword [buffer](#) [8]
- A ring buffer of PWC values.*
- uword [lastCapture](#)
- The last captured value.*
- ubyte [averageOver](#)
- The number of pulses to average over.*
- ubyte [pos](#)
- The current ring position.*
- ubyte [overflow](#)
- The overflow count during the last capture.*
- ubyte [invalid](#)
- This is an invalidation counter.*
- ubyte [state](#)
- The state of the input pin during the last update.*
- } [channels](#) [4]
- Processing data for PWC channels.*

```

• struct {
    ubyte portBit
        The input port configuration bit position.
    ubyte portSel
        The input port configuration bits to select.
    ubyte inBit
        The external interrupt configuration bit position.
    ubyte inSel
        The external interrupt configuration to select.
    ubyte inCount
        The external interrupt configuration bit count.
} hsk_pwc_ports []

```

External input configuration structure.

16.18.1 Detailed Description

HSK Pulse Width Counter implementation.

The Pulse Width Conter (PWC) module uses the T2CCU Capture/Compare Timer (CCT) to measure pulse width.

Author

kami

16.18.2 Macro Definition Documentation

16.18.2.1 BIT_CCM0

```
#define BIT_CCM0 0
```

T2CCU_CCEN Capture/Compare Enable bits start.

16.18.2.2 BIT_CCTBx

```
#define BIT_CCTBx 0
```

T2CCU_CCTBSEL Channel x Time Base Select bit.

16.18.2.3 BIT_CCTOVEN

```
#define BIT_CCTOVEN 2
```

T2CCU_CCTCON Capture/Compare Timer Overflow Interrupt Enable bit.

16.18.2.4 BIT_CCTOVF

```
#define BIT_CCTOVF 3
```

T2CCU_CCTCON Capture/Compare Timer Overflow Flag bit.

16.18.2.5 BIT_CCTPRE

```
#define BIT_CCTPRE 4
```

T2CCU_CCTCON T2CCU Capture/Compare Timer Control Register bits.

16.18.2.6 BIT_CCTST

```
#define BIT_CCTST 0
```

T2CCU_CCTCON Capture/Compare Timer Start/Stop Control bit.

16.18.2.7 BIT_IMODE

```
#define BIT_IMODE 4
```

SYSCON0 Interrupt Structure 2 Mode Select bit.

16.18.2.8 BIT_T2CCFG

```
#define BIT_T2CCFG 4
```

CR_MISC Timer 2 Capture/Compare Unit Clock Configuration bit.

16.18.2.9 BIT_T2CCU_DIS

```
#define BIT_T2CCU_DIS 3
```

PMCON1 T2CCU Disable Request bit.

16.18.2.10 BIT_TIMSYN

```
#define BIT_TIMSYN 1
```

T2CCU_CCTCON Enable synchronized Timer Starts.

16.18.2.11 CHAN_BUF_SIZE

```
#define CHAN_BUF_SIZE 8
```

The size of a PWC ring buffer.

This must not be greater than 32 or the calculation of values returned by [hsk_pwc_channel_getValue\(\)](#) might overflow.

The value 8 should be a sensible compromise between an interest to get averages from a sufficient number of values and memory use.

16.18.2.12 CNT_CCMx

```
#define CNT_CCMx 2
```

CCMx bit count.

16.18.2.13 CNT_EXINTx

```
#define CNT_EXINTx 2
```

EXICONn EXINTx mode bit count.

16.18.2.14 EDGE_DEFAULT_MODE

```
#define EDGE_DEFAULT_MODE PWC_EDGE_BOTH
```

Default to using both edges for pulse detection.

16.18.2.15 PWC_CC0_EXINT_BIT

```
#define PWC_CC0_EXINT_BIT 6
```

The edge detection mode bit position for PWC_CC0.

16.18.2.16 PWC_CC0_EXINT_REG

```
#define PWC_CC0_EXINT_REG EXICON0
```

External Interrupt Control Register for setting the PWC_CC0 edge detection mode.

16.18.2.17 PWC_CC1_EXINT_BIT

```
#define PWC_CC1_EXINT_BIT 0
```

The edge detection mode bit position for PWC_CC1.

16.18.2.18 PWC_CC1_EXINT_REG

```
#define PWC_CC1_EXINT_REG EXICON1
```

External Interrupt Control Register for setting the PWC_CC1 edge detection mode.

16.18.2.19 PWC_CC2_EXINT_BIT

```
#define PWC_CC2_EXINT_BIT 2
```

The edge detection mode bit position for PWC_CC2.

16.18.2.20 PWC_CC2_EXINT_REG

```
#define PWC_CC2_EXINT_REG EXICON1
```

External Interrupt Control Register for setting the PWC_CC2 edge detection mode.

16.18.2.21 PWC_CC3_EXINT_BIT

```
#define PWC_CC3_EXINT_BIT 4
```

The edge detection mode bit position for PWC_CC3.

16.18.2.22 PWC_CC3_EXINT_REG

```
#define PWC_CC3_EXINT_REG EXICON1
```

External Interrupt Control Register for setting the PWC_CC3 edge detection mode.

16.18.2.23 PWC_CHANNELS

```
#define PWC_CHANNELS 4
```

The number of available PWC channels.

16.18.3 Function Documentation

16.18.3.1 hsk_pwc_ccn()

```
void hsk_pwc_ccn (
    const hsk_pwc_channel channel,
    uword capture ) [private]
```

This is the common implementation for soft capture events.

Parameters

<i>channel</i>	The channel that was captured.
<i>capture</i>	The value that was captured.

16.18.3.2 hsk_pwc_channel_captureMode()

```
void hsk_pwc_channel_captureMode (
    const hsk_pwc_channel channel,
    const ubyte captureMode )
```

Allows switching between external and soft trigger.

This does not reconfigure the input ports. Available modes are specified in the PWC_MODE_* defines. PWC_MODE_EXT is the default.

Parameters

<i>channel</i>	The channel to configure.
<i>captureMode</i>	The mode to set the channel to.

16.18.3.3 hsk_pwc_channel_close()

```
void hsk_pwc_channel_close (
    const hsk_pwc_channel channel )
```

Close a PWC channel.

Parameters

<i>channel</i>	The channel to close.
----------------	-----------------------

16.18.3.4 hsk_pwc_channel_edgeMode()

```
void hsk_pwc_channel_edgeMode (
    const hsk_pwc_channel channel,
    const ubyte edgeMode )
```

Select the edge that is used to detect a pulse.

Available edges are specified in the PWC_EDGE_* defines.

Parameters

<i>channel</i>	The channel to configure the edge for.
<i>edgeMode</i>	The selected edge detection mode.

16.18.3.5 hsk_pwc_channel_getValue()

```
ulong hsk_pwc_channel_getValue (
    const hsk_pwc_channel channel,
    const ubyte unit )
```

Returns a measure of the values in a channel buffer.

It also takes care of invalidating channels that haven't been captured for too long.

The value is returned in a requested unit, the units defined as PWC_UNIT_* are available.

Parameters

<i>channel</i>	The channel to return the buffer sum of
<i>unit</i>	The unit to return the channel value in

Return values

<i>>0</i>	The channel value in the requested unit
<i>0</i>	Invalid channel, measurement timed out

16.18.3.6 hsk_pwc_channel_open()

```
void hsk_pwc_channel_open (
    const hsk_pwc_channel channel,
    ubyte averageOver )
```

Configures a PWC channel without an input port.

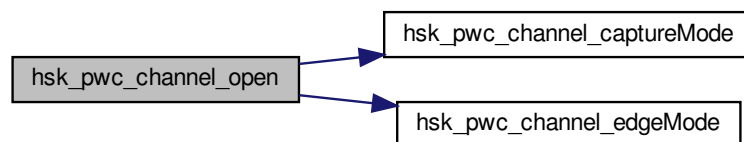
The channel is set up for software triggering (PWC_MODE_SOFT), and triggering on both edges (PWC_EDGE_↔ BOTH).

Parameters

<i>channel</i>	The PWC channel to open
<i>averageOver</i>	The number of pulse values to average over when returning a value or speed. The value must be between 1 and 8.

Set the PWC capture mode.

Set the interrupt triggering edge. Here is the call graph for this function:



16.18.3.7 hsk_pwc_channel_trigger()

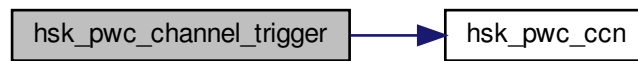
```
void hsk_pwc_channel_trigger (
    const hsk_pwc_channel channel )
```

Triggers a channel in soft trigger mode.

Parameters

<i>channel</i>	The channel to trigger.
----------------	-------------------------

Here is the call graph for this function:



16.18.3.8 hsk_pwc_disable()

```
void hsk_pwc_disable (
    void )
```

Turns off the T2CCU clock to preserve power.

16.18.3.9 hsk_pwc_enable()

```
void hsk_pwc_enable (
    void )
```

Enables T2CCU module if disabled.

16.18.3.10 hsk_pwc_init()

```
void hsk_pwc_init (
    ulong window )
```

This function initializes the T2CCU Capture/Compare Unit for capture mode.

The capturing is based on the CCT timer. Timer T2 is not used and thus can be used without interference.

The window time is the time frame within which pulses should be detected. A smaller time frame results in higher precision, but detection of longer pulses will fail.

Window times vary between $\sim 1\text{ms}$ ($(2^{16} - 1)/(48 * 10^6)$) and $\sim 5592\text{ms}$ ($(2^{16} - 1) * 2^{12}/(48 * 10^6)$). The shortest window time delivers $\sim 20\text{ns}$ and the longest time $\sim 85\mu\text{s}$ precision.

The real window time is on a logarithmic scale (base 2), the init function will select the lowest scale that guarantees the required window time. I.e. the highest precision possible with the desired window time, which is at least 2^{15} for all windows below or equal 5592ms.

Parameters

<i>window</i>	The time in ms to detect a pulse.
---------------	-----------------------------------

Here is the call graph for this function:



16.18.3.11 hsk_pwc_isr_cc0_p30()

```
void hsk_pwc_isr_cc0_p30 (  
    void ) [private]
```

The ISR for Capture events on channel PWC_CC0_P30.

Here is the call graph for this function:



16.18.3.12 hsk_pwc_isr_cc0_p40()

```
void hsk_pwc_isr_cc0_p40 (  
    void ) [private]
```

The ISR for Capture events on channel PWC_CC0_P40.

Here is the call graph for this function:



16.18.3.13 hsk_pwc_isr_cc0_p55()

```
void hsk_pwc_isr_cc0_p55 (  
    void ) [private]
```

The ISR for Capture events on channel PWC_CC0_P55.

Here is the call graph for this function:



16.18.3.14 hsk_pwc_isr_cc1_p32()

```
void hsk_pwc_isr_cc1_p32 (  
    void ) [private]
```

The ISR for Capture events on channel PWC_CC1_P32.

Here is the call graph for this function:



16.18.3.15 hsk_pwc_isr_cc1_p41()

```
void hsk_pwc_isr_cc1_p41 (  
    void ) [private]
```

The ISR for Capture events on channel PWC_CC1_P41.

Here is the call graph for this function:



16.18.3.16 hsk_pwc_isr_cc1_p56()

```
void hsk_pwc_isr_cc1_p56 (  
    void ) [private]
```

The ISR for Capture events on channel PWC_CC1_P56.

Here is the call graph for this function:

**16.18.3.17 hsk_pwc_isr_cc2_p33()**

```
void hsk_pwc_isr_cc2_p33 (  
    void ) [private]
```

The ISR for Capture events on channel PWC_CC2_P33.

Here is the call graph for this function:

**16.18.3.18 hsk_pwc_isr_cc2_p44()**

```
void hsk_pwc_isr_cc2_p44 (  
    void ) [private]
```

The ISR for Capture events on channel PWC_CC2_P44.

Here is the call graph for this function:



16.18.3.19 hsk_pwc_isr_cc2_p52()

```
void hsk_pwc_isr_cc2_p52 (  
    void ) [private]
```

The ISR for Capture events on channel PWC_CC2_P52.

Here is the call graph for this function:



16.18.3.20 hsk_pwc_isr_cc3_p34()

```
void hsk_pwc_isr_cc3_p34 (  
    void ) [private]
```

The ISR for Capture events on channel PWC_CC3_P34.

Here is the call graph for this function:



16.18.3.21 hsk_pwc_isr_cc3_p45()

```
void hsk_pwc_isr_cc3_p45 (  
    void ) [private]
```

The ISR for Capture events on channel PWC_CC3_P45.

Here is the call graph for this function:



16.18.3.22 hsk_pwc_isr_cc3_p57()

```
void hsk_pwc_isr_cc3_p57 (
    void ) [private]
```

The ISR for Capture events on channel PWC_CC3_P57.

Here is the call graph for this function:



16.18.3.23 hsk_pwc_isr_ccn()

```
void hsk_pwc_isr_ccn (
    const hsk_pwc_channel channel,
    uword capture ) [private]
```

This is the common implementation of the Capture ISRs.

Parameters

<i>channel</i>	The channel that was captured.
<i>capture</i>	The value that was captured.

16.18.3.24 hsk_pwc_isr_cctOverflow()

```
void hsk_pwc_isr_cctOverflow (
    void ) [private]
```

The ISR for Capture/Compare overflow events.

It simply increases overflows, which is used by `hsk_pwc_channel_getSum()` to check whether the capture time window was left.

16.18.3.25 hsk_pwc_port_open()

```
void hsk_pwc_port_open (
    const hsk_pwc_port port,
    ubyte averageOver )
```

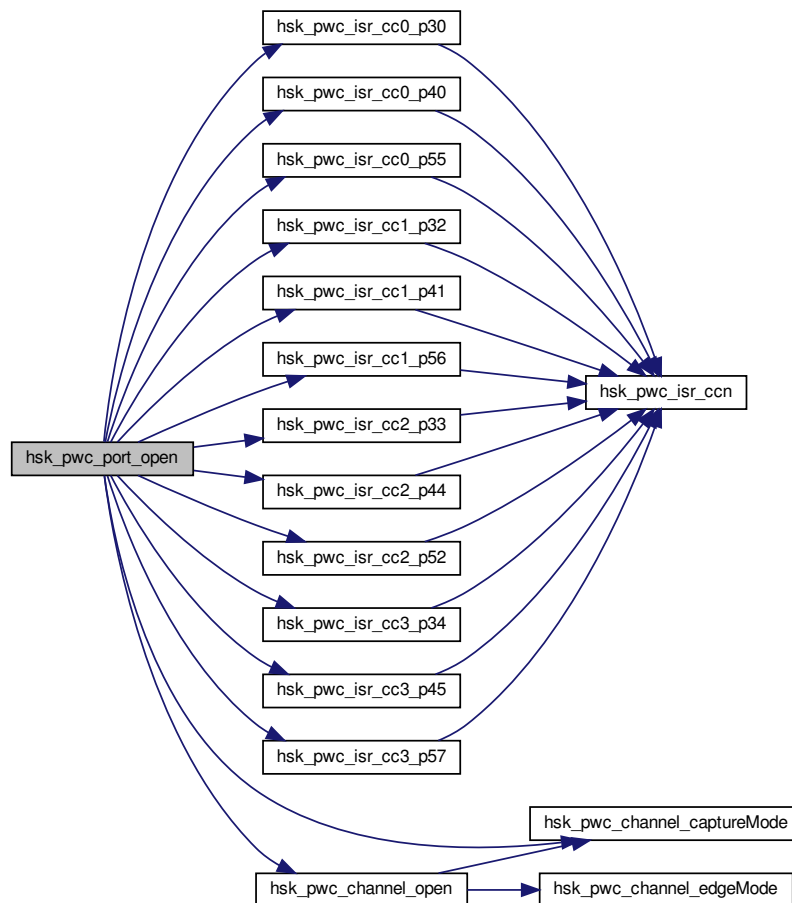
Opens an input port and the connected channel.

The available configurations are available from the `PWC_CCn_*` defines.

Parameters

<i>port</i>	The input port to open
<i>averageOver</i>	The number of pulse values to average over when returning a value or speed. The value must be between 1 and CHAN_BUF_SIZE

Here is the call graph for this function:



16.18.4 Variable Documentation

16.18.4.1 averageOver

```
ubyte averageOver
```

The number of pulses to average over.

16.18.4.2 buffer

```
uword buffer[8]
```

A ring buffer of PWC values.

16.18.4.3 channels

channels [static]

Processing data for PWC channels.

16.18.4.4 hsk_pwc_ports

hsk_pwc_ports

Initial value:

```

= {
    {0, 3, 0, 2, 2},
    {0, 4, 0, 1, 2},
    {5, 2, 0, 3, 2},
    {2, 5, 2, 2, 2},
    {1, 1, 2, 1, 2},
    {6, 2, 2, 3, 2},
    {3, 1, 4, 2, 2},
    {4, 3, 4, 1, 2},
    {2, 2, 4, 3, 2},
    {4, 4, 5, 3, 3},
    {5, 3, 5, 2, 3},
    {7, 3, 5, 4, 3}
}

```

External input configuration structure.

16.18.4.5 inBit

ubyte inBit

The external interrupt configuration bit position.

16.18.4.6 inCount

ubyte inCount

The external interrupt configuration bit count.

16.18.4.7 inSel

ubyte inSel

The external interrupt configuration to select.

16.18.4.8 invalid

ubyte invalid

This is an invalidation counter.

Each time 0 is written into the buffer this is increased. Each time a valid value makes it in it's decreased, so that results are only output by getSum when all values are valid.

16.18.4.9 lastCapture

```
uword lastCapture
```

The last captured value.

16.18.4.10 overflow

```
ubyte overflow
```

The overflow count during the last capture.

This is used by `hsk_pwc_channel_getSum()` to detect whether the capturing time window was left.

16.18.4.11 overflows

```
volatile ubyte overflows [static]
```

A CCT overflow counter.

16.18.4.12 portBit

```
ubyte portBit
```

The input port configuration bit position.

16.18.4.13 portSel

```
ubyte portSel
```

The input port configuration bits to select.

16.18.4.14 pos

```
ubyte pos
```

The current ring position.

16.18.4.15 prescaler

```
ubyte prescaler [static]
```

The prescaling factor.

16.18.4.16 state

```
ubyte state
```

The state of the input pin during the last update.

I.e. in case of 0 a high pulse was completed, in case of 1 a low pulse.

16.18.4.17 sum

```
ulong sum
```

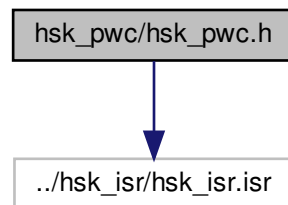
The sum of the values stored in the ring buffer.

16.19 hsk_pwc/hsk_pwc.h File Reference

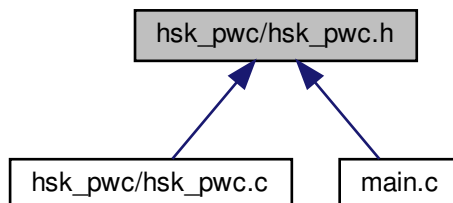
HSK Pulse Width Counter headers.

```
#include "../hsk_isr/hsk_isr.isr"
```

Include dependency graph for hsk_pwc.h:



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



Macros

- `#define PWC_CC0 0`
Capture/Compare channel 0 on EXINT3.
- `#define PWC_CC1 1`
Capture/Compare channel 1 on EXINT4.
- `#define PWC_CC2 2`
Capture/Compare channel 2 on EXINT5.
- `#define PWC_CC3 3`
Capture/Compare channel 3 on EXINT6.
- `#define PWC_CC0_P30 0`
Capture/Compare channel 0 input port P3.0 configuration.
- `#define PWC_CC0_P40 1`
Capture/Compare channel 0 input port P4.0 configuration.
- `#define PWC_CC0_P55 2`
Capture/Compare channel 0 input port P5.5 configuration.
- `#define PWC_CC1_P32 3`
Capture/Compare channel 1 input port P3.2 configuration.
- `#define PWC_CC1_P41 4`
Capture/Compare channel 1 input port P4.1 configuration.
- `#define PWC_CC1_P56 5`
Capture/Compare channel 1 input port P5.6 configuration.
- `#define PWC_CC2_P33 6`
Capture/Compare channel 2 input port P3.3 configuration.
- `#define PWC_CC2_P44 7`
Capture/Compare channel 4 input port P4.4 configuration.
- `#define PWC_CC2_P52 8`
Capture/Compare channel 2 input port P5.2 configuration.
- `#define PWC_CC3_P34 9`
Capture/Compare channel 3 input port P3.4 configuration.
- `#define PWC_CC3_P45 10`
Capture/Compare channel 3 input port P4.5 configuration.
- `#define PWC_CC3_P57 11`
Capture/Compare channel 3 input port P5.7 configuration.
- `#define PWC_EDGE_FALLING 0`
Configuration selection to trigger pulse detection on falling edge.
- `#define PWC_EDGE_RISING 1`
Configuration selection to trigger pulse detection on rising edge.
- `#define PWC_EDGE_BOTH 2`
Configuration selection to trigger pulse detection on both edges.
- `#define PWC_MODE_EXT 1`
Available capture modes, capture on external interrupt.
- `#define PWC_MODE_SOFT 3`
Available capture modes, capture on software event.
- `#define PWC_UNIT_SUM_RAW 0`
*Sum of buffered pulse widths in multiples of $1/48 * 10^{-6} s$.*
- `#define PWC_UNIT_WIDTH_RAW 1`
*Average of buffered pulse widths in multiples of $1/48 * 10^{-6} s$.*
- `#define PWC_UNIT_WIDTH_NS 2`
Average of buffered pulse widths in multiples of $10^{-9} s$.
- `#define PWC_UNIT_WIDTH_US 3`

- Average of buffered pulse widths in multiples of $10^{-6} s$.*

 - `#define PWC_UNIT_WIDTH_MS` 4
- Average of buffered pulse widths in multiples of $10^{-3} s$.*

 - `#define PWC_UNIT_FREQ_S` 5
- Average frequency of buffered pulses in multiples of $1/s$.*

 - `#define PWC_UNIT_FREQ_M` 6
- Average frequency of buffered pulses in multiples of $1/m$.*

 - `#define PWC_UNIT_FREQ_H` 7
- Average frequency of buffered pulses in multiples of $1/h$.*

 - `#define PWC_UNIT_DUTYH_RAW` 8
- Latest high pulse in multiples of $1/48 * 10^{-6} s$.*

 - `#define PWC_UNIT_DUTYH_NS` 9
- Latest high pulse in multiples of $1 * 10^{-9} s$.*

 - `#define PWC_UNIT_DUTYH_US` 10
- Latest high pulse in multiples of $1 * 10^{-6} s$.*

 - `#define PWC_UNIT_DUTYH_MS` 11
- Latest high pulse in multiples of $1 * 10^{-3} s$.*

 - `#define PWC_UNIT_DUTYL_RAW` 12
- Latest low pulse in multiples of $1/48 * 10^{-6} s$.*

 - `#define PWC_UNIT_DUTYL_NS` 13
- Latest low pulse in multiples of $1 * 10^{-9} s$.*

 - `#define PWC_UNIT_DUTYL_US` 14
- Latest low pulse in multiples of $1 * 10^{-6} s$.*

 - `#define PWC_UNIT_DUTYL_MS` 15
- Latest low pulse in multiples of $1 * 10^{-3} s$.*

Typedefs

- typedef ubyte `hsk_pwc_channel`
Typedef for PWC channel IDs.
- typedef ubyte `hsk_pwc_port`
Typedef for PWC input port.

Functions

- void `hsk_pwc_init` (ulong window)
This function initializes the T2CCU Capture/Compare Unit for capture mode.
- void `hsk_pwc_channel_open` (const `hsk_pwc_channel` channel, ubyte `averageOver`)
Configures a PWC channel without an input port.
- void `hsk_pwc_port_open` (const `hsk_pwc_port` port, ubyte `averageOver`)
Opens an input port and the connected channel.
- void `hsk_pwc_channel_close` (const `hsk_pwc_channel` channel)
Close a PWC channel.
- void `hsk_pwc_channel_edgeMode` (const `hsk_pwc_channel` channel, const ubyte edgeMode)
Select the edge that is used to detect a pulse.
- void `hsk_pwc_channel_captureMode` (const `hsk_pwc_channel` channel, const ubyte captureMode)
Allows switching between external and soft trigger.
- void `hsk_pwc_channel_trigger` (const `hsk_pwc_channel` channel)
Triggers a channel in soft trigger mode.

- void [hsk_pwc_enable](#) (void)
Enables T2CCU module if disabled.
- void [hsk_pwc_disable](#) (void)
Turns off the T2CCU clock to preserve power.
- ulong [hsk_pwc_channel_getValue](#) (const [hsk_pwc_channel](#) channel, const ubyte unit)
Returns a measure of the values in a channel buffer.

16.19.1 Detailed Description

HSK Pulse Width Counter headers.

This library uses the T2CCU to measure pulse width on the external interrupt pins.

Every caputre channel blocks an external interrupt. Opening a channel will block this interrupt and change its configuration.

Pulse with measurement has a window time that is configured with [hsk_pwc_init\(\)](#) and defines the time frame within which pulses can be detected.

If no pulse occurs during the window, the channel buffer is invalidated and the [hsk_pwc_channel_getValue\(\)](#) function will returns invalid (0) until the buffer is repopulated with valid measurements.

In order to guarantee the detection of invalid channels, the [hsk_pwc_channel_getValue\(\)](#) function has to be called at least once every 256 window times.

Author

kami

16.19.2 Macro Definition Documentation

16.19.2.1 PWC_CC0

```
#define PWC_CC0 0
```

Capture/Compare channel 0 on EXINT3.

16.19.2.2 PWC_CC0_P30

```
#define PWC_CC0_P30 0
```

Capture/Compare channel 0 input port P3.0 configuration.

16.19.2.3 PWC_CC0_P40

```
#define PWC_CC0_P40 1
```

Capture/Compare channel 0 input port P4.0 configuration.

16.19.2.4 PWC_CC0_P55

```
#define PWC_CC0_P55 2
```

Capture/Compare channel 0 input port P5.5 configuration.

16.19.2.5 PWC_CC1

```
#define PWC_CC1 1
```

Capture/Compare channel 1 on EXINT4.

16.19.2.6 PWC_CC1_P32

```
#define PWC_CC1_P32 3
```

Capture/Compare channel 1 input port P3.2 configuration.

16.19.2.7 PWC_CC1_P41

```
#define PWC_CC1_P41 4
```

Capture/Compare channel 1 input port P4.1 configuration.

16.19.2.8 PWC_CC1_P56

```
#define PWC_CC1_P56 5
```

Capture/Compare channel 1 input port P5.6 configuration.

16.19.2.9 PWC_CC2

```
#define PWC_CC2 2
```

Capture/Compare channel 2 on EXINT5.

16.19.2.10 PWC_CC2_P33

```
#define PWC_CC2_P33 6
```

Capture/Compare channel 2 input port P3.3 configuration.

16.19.2.11 PWC_CC2_P44

```
#define PWC_CC2_P44 7
```

Capture/Compare channel 4 input port P4.4 configuration.

16.19.2.12 PWC_CC2_P52

```
#define PWC_CC2_P52 8
```

Capture/Compare channel 2 input port P5.2 configuration.

16.19.2.13 PWC_CC3

```
#define PWC_CC3 3
```

Capture/Compare channel 3 on EXINT6.

16.19.2.14 PWC_CC3_P34

```
#define PWC_CC3_P34 9
```

Capture/Compare channel 3 input port P3.4 configuration.

16.19.2.15 PWC_CC3_P45

```
#define PWC_CC3_P45 10
```

Capture/Compare channel 3 input port P4.5 configuration.

16.19.2.16 PWC_CC3_P57

```
#define PWC_CC3_P57 11
```

Capture/Compare channel 3 input port P5.7 configuration.

16.19.2.17 PWC_EDGE_BOTH

```
#define PWC_EDGE_BOTH 2
```

Configuration selection to trigger pulse detection on both edges.

16.19.2.18 PWC_EDGE_FALLING

```
#define PWC_EDGE_FALLING 0
```

Configuration selection to trigger pulse detection on falling edge.

16.19.2.19 PWC_EDGE_RISING

```
#define PWC_EDGE_RISING 1
```

Configuration selection to trigger pulse detection on rising edge.

16.19.2.20 PWC_MODE_EXT

```
#define PWC_MODE_EXT 1
```

Available capture modes, capture on external interrupt.

16.19.2.21 PWC_MODE_SOFT

```
#define PWC_MODE_SOFT 3
```

Available capture modes, capture on software event.

16.19.3 Typedef Documentation

16.19.3.1 hsk_pwc_channel

```
typedef ubyte hsk_pwc_channel
```

Typedef for PWC channel IDs.

16.19.3.2 hsk_pwc_port

```
typedef ubyte hsk_pwc_port
```

Typedef for PWC input port.

16.19.4 Function Documentation

16.19.4.1 hsk_pwc_channel_captureMode()

```
void hsk_pwc_channel_captureMode (
    const hsk_pwc_channel channel,
    const ubyte captureMode )
```

Allows switching between external and soft trigger.

This does not reconfigure the input ports. Available modes are specified in the PWC_MODE_* defines. PWC_MODE_EXT is the default.

Parameters

<i>channel</i>	The channel to configure.
<i>captureMode</i>	The mode to set the channel to.

16.19.4.2 hsk_pwc_channel_close()

```
void hsk_pwc_channel_close (
    const hsk_pwc_channel channel )
```

Close a PWC channel.

Parameters

<i>channel</i>	The channel to close.
----------------	-----------------------

16.19.4.3 hsk_pwc_channel_edgeMode()

```
void hsk_pwc_channel_edgeMode (
    const hsk_pwc_channel channel,
    const ubyte edgeMode )
```

Select the edge that is used to detect a pulse.

Available edges are specified in the PWC_EDGE_* defines.

Parameters

<i>channel</i>	The channel to configure the edge for.
<i>edgeMode</i>	The selected edge detection mode.

16.19.4.4 hsk_pwc_channel_getValue()

```
ulong hsk_pwc_channel_getValue (
    const hsk_pwc_channel channel,
    const ubyte unit )
```

Returns a measure of the values in a channel buffer.

It also takes care of invalidating channels that haven't been captured for too long.

The value is returned in a requested unit, the units defined as PWC_UNIT_* are available.

Parameters

<i>channel</i>	The channel to return the buffer sum of
<i>unit</i>	' The unit to return the channel value in

Return values

<i>>0</i>	The channel value in the requested unit
<i>0</i>	Invalid channel, measurement timed out

16.19.4.5 hsk_pwc_channel_open()

```
void hsk_pwc_channel_open (
    const hsk_pwc_channel channel,
    ubyte averageOver )
```

Configures a PWC channel without an input port.

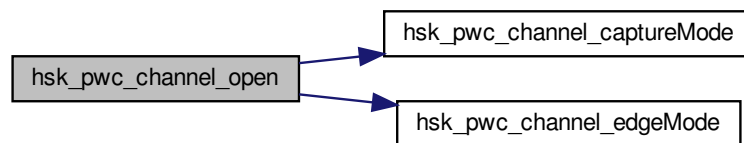
The channel is set up for software triggering (PWC_MODE_SOFT), and triggering on both edges (PWC_EDGE_↔BOTH).

Parameters

<i>channel</i>	The PWC channel to open
<i>averageOver</i>	The number of pulse values to average over when returning a value or speed. The value must be between 1 and 8.

Set the PWC capture mode.

Set the interrupt triggering edge. Here is the call graph for this function:



16.19.4.6 hsk_pwc_channel_trigger()

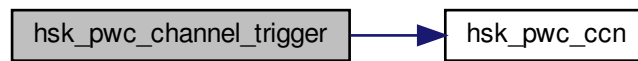
```
void hsk_pwc_channel_trigger (
    const hsk_pwc_channel channel )
```

Triggers a channel in soft trigger mode.

Parameters

<i>channel</i>	The channel to trigger.
----------------	-------------------------

Here is the call graph for this function:



16.19.4.7 hsk_pwc_disable()

```
void hsk_pwc_disable (
    void )
```

Turns off the T2CCU clock to preserve power.

16.19.4.8 hsk_pwc_enable()

```
void hsk_pwc_enable (
    void )
```

Enables T2CCU module if disabled.

16.19.4.9 hsk_pwc_init()

```
void hsk_pwc_init (
    ulong window )
```

This function initializes the T2CCU Capture/Compare Unit for capture mode.

The capturing is based on the CCT timer. Timer T2 is not used and thus can be used without interference.

The window time is the time frame within which pulses should be detected. A smaller time frame results in higher precision, but detection of longer pulses will fail.

Window times vary between $\sim 1\text{ms}$ ($(2^{16} - 1)/(48 * 10^6)$) and $\sim 5592\text{ms}$ ($(2^{16} - 1) * 2^{12}/(48 * 10^6)$). The shortest window time delivers $\sim 20\text{ns}$ and the longest time $\sim 85\mu\text{s}$ precision.

The real window time is on a logarithmic scale (base 2), the init function will select the lowest scale that guarantees the required window time. I.e. the highest precision possible with the desired window time, which is at least 2^{15} for all windows below or equal 5592ms.

Parameters

<i>window</i>	The time in ms to detect a pulse.
---------------	-----------------------------------

Here is the call graph for this function:



16.19.4.10 hsk_pwc_port_open()

```
void hsk_pwc_port_open (
    const hsk_pwc_port port,
    ubyte averageOver )
```

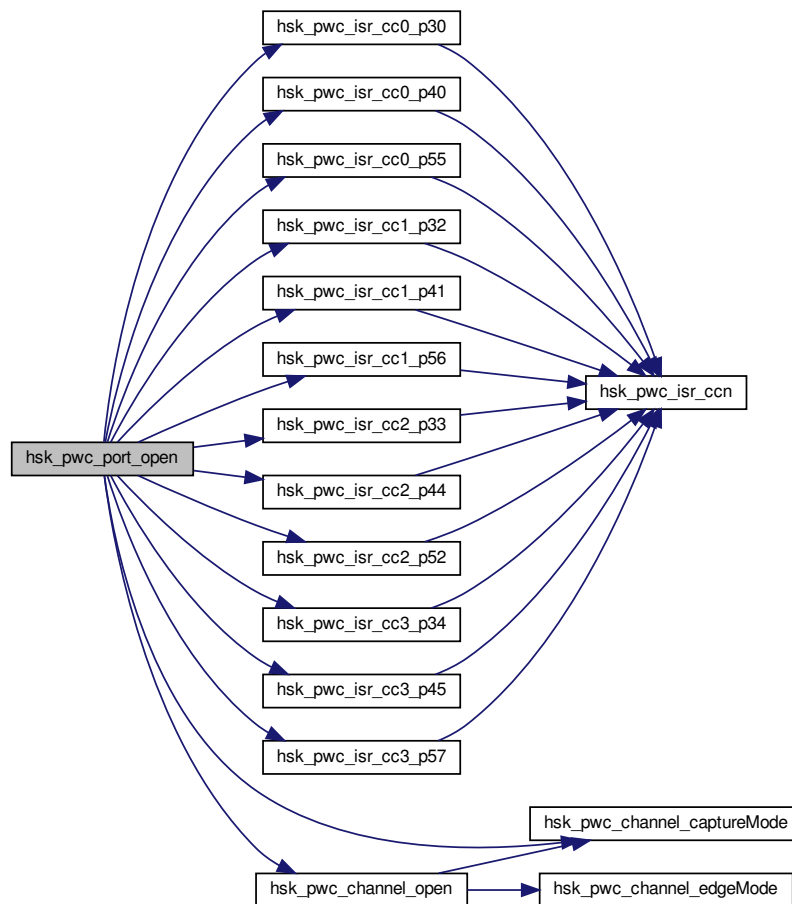
Opens an input port and the connected channel.

The available configurations are available from the PWC_CCn_* defines.

Parameters

<i>port</i>	The input port to open
<i>averageOver</i>	The number of pulse values to average over when returning a value or speed. The value must be between 1 and CHAN_BUF_SIZE

Here is the call graph for this function:

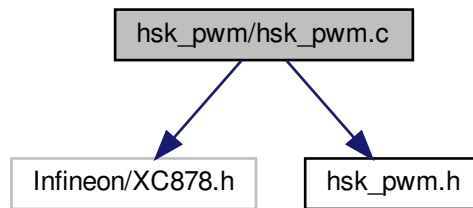


16.20 hsk_pwm/hsk_pwm.c File Reference

HSK Pulse Width Modulation implementation.

```
#include <Infineon/XC878.h>
#include "hsk_pwm.h"
```

Include dependency graph for hsk_pwm.c:



Macros

- #define [BIT_CCUCCFG](#) 5
CR_MISC CCU6 Clock Configuration bit.
- #define [BIT_TnCLK](#) 0
CCU6_TCTR0L/CCU6_TCTR0H Timer T12/T13 Input Clock Select and Prescaler bits.
- #define [CNT_TnCLK](#) 4
TnCLK bit count.
- #define [BIT_PSL](#) 0
PSLR Compare Outputs Passive State Level bits.
- #define [CNT_PSL](#) 6
PSL bit count.
- #define [BIT_PSL63](#) 7
PSLR Passive State Level of Output COUT63 bit.
- #define [BIT_TnMODEN](#) 0
CCU6_MODCTRL/CCU6_MODCTRH T12/T13 Modulation Enable bits.
- #define [CNT_TnMODEN](#) 6
TnMODEN bit count.
- #define [BIT_ECT13O](#) 7
CCU6_MODCTRH Enable Compare Timer T13 Output bits.
- #define [CNT_MSEL6n](#) 4
T12MSELL/H Capture/Compare Mode Selection width.
- #define [MOD_MSEL6n](#) 0x3
T12MSELL/H Capture/Compare Mode Selection mode.
- #define [BIT_TnSTR](#) 6
CCU6_TCTR4L/CCU6_TCTR4H Timer T12/T13 Shadow Transfer Request bit.
- #define [BIT_CCU_DIS](#) 2
PMCON1 Capture Compare Unit Disable bit.
- #define [BIT_TnRRR](#) 0
CCU6_TCTR4L/CCU6_TCTR4H Timer T12/T13 Run Reset bit.
- #define [BIT_TnRS](#) 1
CCU6_TCTR4L/CCU6_TCTR4H Timer T12/T13 Run Set bit.

Functions

- void `hsk_pwm_init` (const `hsk_pwm_channel` channel, const `ulong` freq)
Sets up the the CCU6 timer frequencies that control the PWM cycle.
- void `hsk_pwm_port_open` (const `hsk_pwm_port` port)
Set up a PWM output port.
- void `hsk_pwm_port_close` (const `hsk_pwm_port` port)
Close a PWM output port.
- void `hsk_pwm_channel_set` (const `hsk_pwm_channel` channel, const `uword` max, const `uword` value)
Set the duty cycle for the given channel.
- void `hsk_pwm_outChannel_dir` (`hsk_pwm_outChannel` channel, const `bool` up)
Set the direction of an output channel.
- void `hsk_pwm_enable` (void)
Turns on the CCU6.
- void `hsk_pwm_disable` (void)
Deactivates the CCU6 to reduce power consumption.

Variables

- struct {
 `ubyte` `pos`
 The Pn_ALTSEL[01] bit position to make the port configuration in.
 `ubyte` `sel`
 Select a 2 bits Pn_ALTSEL[01] configuration.
} `ports` []

Data structure to hold output port configurations.

16.20.1 Detailed Description

HSK Pulse Width Modulation implementation.

This would mostly be straightforward if it wasn't for the messy output channel configuration.

The init function buys a lot of simplicity by limiting the CCU6 use to generating PWM. Also, the channels PWM_60, PWM_61 and PWM_62 operate at the same base frequency and period. This is a hardware limitation.

Author

kami

16.20.2 Macro Definition Documentation

16.20.2.1 BIT_CCU_DIS

```
#define BIT_CCU_DIS 2
```

PMCON1 Capture Compare Unit Disable bit.

16.20.2.2 BIT_CCUCCFG

```
#define BIT_CCUCCFG 5
```

CR_MISC CCU6 Clock Configuration bit.

16.20.2.3 BIT_ECT130

```
#define BIT_ECT130 7
```

CCU6_MODCTRH Enable Compare Timer T13 Output bits.

16.20.2.4 BIT_PSL

```
#define BIT_PSL 0
```

PSLR Compare Outputs Passive State Level bits.

16.20.2.5 BIT_PSL63

```
#define BIT_PSL63 7
```

PSLR Passive State Level of Output COUT63 bit.

16.20.2.6 BIT_TnCLK

```
#define BIT_TnCLK 0
```

CCU6_TCTR0L/CCU6_TCTR0H Timer T12/T13 Input Clock Select and Prescaler bits.

16.20.2.7 BIT_TnMODEN

```
#define BIT_TnMODEN 0
```

CCU6_MODCTRL/CCU6_MODCTRH T12/T13 Modulation Enable bits.

16.20.2.8 BIT_TnRR

```
#define BIT_TnRR 0
```

CCU6_TCTR4L/CCU6_TCTR4H Timer T12/T13 Run Reset bit.

16.20.2.9 BIT_TnRS

```
#define BIT_TnRS 1
```

CCU6_TCTR4L/CCU6_TCTR4H Timer T12/T13 Run Set bit.

16.20.2.10 BIT_TnSTR

```
#define BIT_TnSTR 6
```

CCU6_TCTR4L/CCU6_TCTR4H Timer T12/T13 Shadow Transfer Request bit.

16.20.2.11 CNT_MSEL6n

```
#define CNT_MSEL6n 4
```

T12MSELL/H Capture/Compare Mode Selection width.

16.20.2.12 CNT_PSL

```
#define CNT_PSL 6
```

PSL bit count.

16.20.2.13 CNT_TnCLK

```
#define CNT_TnCLK 4
```

TnCLK bit count.

16.20.2.14 CNT_TnMODEN

```
#define CNT_TnMODEN 6
```

TnMODEN bit count.

16.20.2.15 MOD_MSEL6n

```
#define MOD_MSEL6n 0x3
```

T12MSELL/H Capture/Compare Mode Selection mode.

This mode means CC6n and COUT6n are in output mode.

16.20.3 Function Documentation**16.20.3.1 hsk_pwm_channel_set()**

```
void hsk_pwm_channel_set (
    const hsk_pwm_channel channel,
    const uword max,
    const uword value )
```

Set the duty cycle for the given channel.

I.e. the active time frame slice of period can be set with max and value.

To set the duty cycle in percent specify a max of 100 and values from 0 to 100.

Parameters

<i>channel</i>	The PWM channel to set the duty cycle for, check the PWM_6x defines
<i>max</i>	Defines the scope value can move in
<i>value</i>	The current duty cycle value

16.20.3.2 hsk_pwm_disable()

```
void hsk_pwm_disable (
    void )
```

Deactivates the CCU6 to reduce power consumption.

16.20.3.3 hsk_pwm_enable()

```
void hsk_pwm_enable (
    void )
```

Turns on the CCU6.

Deactivates the power disable mode and sets the T12 and T13 Timer Run bits.

Precondition

All [hsk_pwm_init\(\)](#) calls have to be completed to call this

16.20.3.4 hsk_pwm_init()

```
void hsk_pwm_init (
    const hsk_pwm_channel channel,
    const ulong freq )
```

Sets up the the CCU6 timer frequencies that control the PWM cycle.

The channels PWM_60, PWM_61 and PWM_62 share the timer T12, thus initializing one of them, initializes them all. The channel PWM_63 has exclusive use of the timer T13 and can thus be used with its own operating frequency.

Frequencies up to ~732.4Hz are always between 15 and 16 bits precision.

Frequencies above 48kHz offer less than 1/1000 precision. From there it is a linear function, i.e. 480kHz still offer 1/100 precision.

The freq value 0 will result in ~0.02Hz ($48000000/2^{31}$).

The following formula results in the freq value that yields exactly the desired precision, this is useful to avoid precision loss by rounding:

$$freq(precision) = 480000000 * precision$$

E.g. 10 bit precision: $freq(1/2^{10}) = 468750$

Parameters

<i>channel</i>	The channel to change the frequency for
<i>freq</i>	The desired PWM cycle frequency in units of 0.1Hz

PWM Timings

The CCU6CLK can run at FCLK (48MHz) or PCLK (24MHz), configured in the CCUCCFG bit. This implementation always uses 48MHz.

The T12CLK can run any power of two between CCU6CLK and CCU6CLK/128, configured in the T12CLK bit field.

This value can additionally be multiplied with a prescaler of 1/256, activated with the T12PRE bit.

The same is true for the T13CLK.

Additionally the period is length for T12 and T13 can be configured to any 16 bit value. Assuming at least 1/1000 precision is desired that means the clock cycle can be shortened by any factor up to 2^6 (64).

The conclusion is that PWM frequencies between 48kHz and $\sim 0.02\text{Hz}$ can be configured. Very high values degrade the precision, e.g. 96kHz will only offer 1/500 precision. The freq value 0 will result in $\sim 0.02\text{Hz}$ ($48000000/2^{31}$).

16.20.3.5 hsk_pwm_outChannel_dir()

```
void hsk_pwm_outChannel_dir (
    hsk_pwm_outChannel channel,
    const bool up )
```

Set the direction of an output channel.

The channel value can be taken from any of the PWM_CCx/PWM_COUTx defines.

Parameters

<i>channel</i>	The IO channel to set the direction bit for
<i>up</i>	Set 1 to output a 1 during the cycle set with hsk_pwm_channel_set() , set 0 to output a 0 during the cycle set with hsk_pwm_channel_set()

16.20.3.6 hsk_pwm_port_close()

```
void hsk_pwm_port_close (
    const hsk_pwm_port port )
```

Close a PWM output port.

This configures the necessary port direction bits.

The port can be any one of the PWM_OUT_x_* defines.

Parameters

<i>port</i>	The output port to deactivate
-------------	-------------------------------

16.20.3.7 hsk_pwm_port_open()

```
void hsk_pwm_port_open (
    const hsk_pwm_port port )
```

Set up a PWM output port.

This configures the necessary port direction bits and activates the corresponding output channels.

The port can be any one of the PWM_OUT_x_* defines.

Precondition

This function should only be called after [hsk_pwm_enable\(\)](#), otherwise the output port will be driven (1) until PWM is enabled

Parameters

<i>port</i>	The output port to activate
-------------	-----------------------------

16.20.4 Variable Documentation

16.20.4.1 ports

```
ports [static]
```

Initial value:

```
= {
    {0, 1},
    {1, 1},
    {0, 1},
    {1, 1},
    {0, 2},
    {1, 2},
    {1, 2},
    {2, 1},
    {3, 1},
    {4, 1},
    {5, 1},
    {4, 2},
    {5, 2},
    {4, 1},
    {5, 1},
    {6, 1},
    {7, 1},
    {3, 2},
    {7, 1},
    {3, 2}
}
```

Data structure to hold output port configurations.

16.20.4.2 pos

```
ubyte pos
```

The Pn_ALTSEL[01] bit position to make the port configuration in.

16.20.4.3 sel

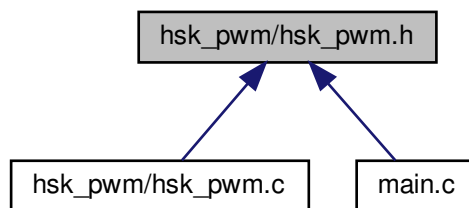
```
ubyte sel
```

Select a 2 bits Pn_ALTSEL[01] configuration.

16.21 hsk_pwm/hsk_pwm.h File Reference

HSK Pulse Width Modulation headers.

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



Macros

- `#define PWM_60 0`
PWM channel 60, Timer T12 driven.
- `#define PWM_61 1`
PWM channel 61, Timer T12 driven.
- `#define PWM_62 2`
PWM channel 62, Timer T12 driven.
- `#define PWM_63 3`
PWM channel 63, Timer T13 driven.
- `#define PWM_CC60 0`
IO channel configuration for PWM_60.
- `#define PWM_COUT60 1`
Output channel configuration for PWM_60.
- `#define PWM_CC61 2`
IO channel configuration for PWM_61.
- `#define PWM_COUT61 3`

- Output channel configuration for PWM_61.*
- #define [PWM_CC62](#) 4
 - IO channel configuration for PWM_62.*
- #define [PWM_COUT62](#) 5
 - Output channel configuration for PWM_62,.*
- #define [PWM_COUT63](#) 6
 - Output channel configuration for PWM_63.*
- #define [PWM_OUT_60_P30](#) 0
 - PWM_60 output configuration for P3.0 through PWM_CC60.*
- #define [PWM_OUT_60_P31](#) 1
 - PWM_60 output configuration for P3.1 through PWM_COUT60.*
- #define [PWM_OUT_60_P40](#) 2
 - PWM_60 output configuration for P4.0 through PWM_CC60.*
- #define [PWM_OUT_60_P41](#) 3
 - PWM_60 output configuration for P4.1 through PWM_COUT60.*
- #define [PWM_OUT_61_P00](#) 4
 - PWM_61 output configuration for P0.0 through PWM_CC61.*
- #define [PWM_OUT_61_P01](#) 5
 - PWM_61 output configuration for P0.1 through PWM_COUT61.*
- #define [PWM_OUT_61_P31](#) 6
 - PWM_61 output configuration for P3.1 through PWM_CC61.*
- #define [PWM_OUT_61_P32](#) 7
 - PWM_61 output configuration for P3.2 through PWM_CC61.*
- #define [PWM_OUT_61_P33](#) 8
 - PWM_61 output configuration for P3.3 through PWM_COUT61.*
- #define [PWM_OUT_61_P44](#) 9
 - PWM_61 output configuration for P4.4 through PWM_CC61.*
- #define [PWM_OUT_61_P45](#) 10
 - PWM_61 output configuration for P4.5 through PWM_COUT61.*
- #define [PWM_OUT_62_P04](#) 11
 - PWM_62 output configuration for P0.4 through PWM_CC62.*
- #define [PWM_OUT_62_P05](#) 12
 - PWM_62 output configuration for P0.5 through PWM_COUT62.*
- #define [PWM_OUT_62_P34](#) 13
 - PWM_62 output configuration for P3.4 through PWM_CC62.*
- #define [PWM_OUT_62_P35](#) 14
 - PWM_62 output configuration for P3.5 through PWM_COUT62.*
- #define [PWM_OUT_62_P46](#) 15
 - PWM_62 output configuration for P4.6 through PWM_CC62.*
- #define [PWM_OUT_62_P47](#) 16
 - PWM_62 output configuration for P4.7 through PWM_COUT62.*
- #define [PWM_OUT_63_P03](#) 17
 - PWM_63 output configuration for P0.3 through PWM_COUT63.*
- #define [PWM_OUT_63_P37](#) 18
 - PWM_63 output configuration for P3.7 through PWM_COUT63.*
- #define [PWM_OUT_63_P43](#) 19
 - PWM_63 output configuration for P4.3 through PWM_COUT63.*

Typedefs

- typedef ubyte [hsk_pwm_channel](#)
Type definition for PWM channels.
- typedef ubyte [hsk_pwm_outChannel](#)
Type definition for output channels.
- typedef ubyte [hsk_pwm_port](#)
Type definition for ports.

Functions

- void [hsk_pwm_init](#) (const [hsk_pwm_channel](#) channel, const ulong freq)
Sets up the the CCU6 timer frequencies that control the PWM cycle.
- void [hsk_pwm_port_open](#) (const [hsk_pwm_port](#) port)
Set up a PWM output port.
- void [hsk_pwm_port_close](#) (const [hsk_pwm_port](#) port)
Close a PWM output port.
- void [hsk_pwm_channel_set](#) (const [hsk_pwm_channel](#) channel, const uword max, const uword value)
Set the duty cycle for the given channel.
- void [hsk_pwm_outChannel_dir](#) ([hsk_pwm_outChannel](#) channel, const bool up)
Set the direction of an output channel.
- void [hsk_pwm_enable](#) (void)
Turns on the CCU6.
- void [hsk_pwm_disable](#) (void)
Deactivates the CCU6 to reduce power consumption.

16.21.1 Detailed Description

HSK Pulse Width Modulation headers.

This file provides function prototypes to perform Timer T12 and T13 based PWM with CCU6.

The CCU6 offers the following PWM channels:

- PWM_60
- PWM_61
- PWM_62
- PWM_63

Each PWM channel is connected to two IO channels for output:

- PWM_CCx
- PWM_COUTx

The distinction between PWM and IO channels is important to understand the side effects of some operations.

Refer to the PWM_OUT_x_* defines to know which channel can be connected to which output pins.

The functions are implemented under the assumption, that the use of the timers T12 and T13 as well of the CCU6 is exclusive to this library.

The safe boot order for pwm output is the following:

- [hsk_pwm_init\(\)](#)
- [hsk_pwm_enable\(\)](#)
- [hsk_pwm_port_open\(\)](#)

Author

kami

16.21.2 Macro Definition Documentation

16.21.2.1 PWM_60

```
#define PWM_60 0
```

PWM channel 60, Timer T12 driven.

16.21.2.2 PWM_61

```
#define PWM_61 1
```

PWM channel 61, Timer T12 driven.

16.21.2.3 PWM_62

```
#define PWM_62 2
```

PWM channel 62, Timer T12 driven.

16.21.2.4 PWM_63

```
#define PWM_63 3
```

PWM channel 63, Timer T13 driven.

16.21.2.5 PWM_CC60

```
#define PWM_CC60 0
```

IO channel configuration for PWM_60.

16.21.2.6 PWM_CC61

```
#define PWM_CC61 2
```

IO channel configuration for PWM_61.

16.21.2.7 PWM_CC62

```
#define PWM_CC62 4
```

IO channel configuration for PWM_62.

16.21.2.8 PWM_COUT60

```
#define PWM_COUT60 1
```

Output channel configuration for PWM_60.

16.21.2.9 PWM_COUT61

```
#define PWM_COUT61 3
```

Output channel configuration for PWM_61.

16.21.2.10 PWM_COUT62

```
#define PWM_COUT62 5
```

Output channel configuration for PWM_62,.

16.21.2.11 PWM_COUT63

```
#define PWM_COUT63 6
```

Output channel configuration for PWM_63.

16.21.2.12 PWM_OUT_60_P30

```
#define PWM_OUT_60_P30 0
```

PWM_60 output configuration for P3.0 through PWM_CC60.

16.21.2.13 PWM_OUT_60_P31

```
#define PWM_OUT_60_P31 1
```

PWM_60 output configuration for P3.1 through PWM_COUT60.

16.21.2.14 PWM_OUT_60_P40

```
#define PWM_OUT_60_P40 2
```

PWM_60 output configuration for P4.0 through PWM_CC60.

16.21.2.15 PWM_OUT_60_P41

```
#define PWM_OUT_60_P41 3
```

PWM_60 output configuration for P4.1 through PWM_COUT60.

16.21.2.16 PWM_OUT_61_P00

```
#define PWM_OUT_61_P00 4
```

PWM_61 output configuration for P0.0 through PWM_CC61.

16.21.2.17 PWM_OUT_61_P01

```
#define PWM_OUT_61_P01 5
```

PWM_61 output configuration for P0.1 through PWM_COUT61.

16.21.2.18 PWM_OUT_61_P31

```
#define PWM_OUT_61_P31 6
```

PWM_61 output configuration for P3.1 through PWM_CC61.

16.21.2.19 PWM_OUT_61_P32

```
#define PWM_OUT_61_P32 7
```

PWM_61 output configuration for P3.2 through PWM_CC61.

16.21.2.20 PWM_OUT_61_P33

```
#define PWM_OUT_61_P33 8
```

PWM_61 output configuration for P3.3 through PWM_COUT61.

16.21.2.21 PWM_OUT_61_P44

```
#define PWM_OUT_61_P44 9
```

PWM_61 output configuration for P4.4 through PWM_CC61.

16.21.2.22 PWM_OUT_61_P45

```
#define PWM_OUT_61_P45 10
```

PWM_61 output configuration for P4.5 through PWM_COUT61.

16.21.2.23 PWM_OUT_62_P04

```
#define PWM_OUT_62_P04 11
```

PWM_62 output configuration for P0.4 through PWM_CC62.

16.21.2.24 PWM_OUT_62_P05

```
#define PWM_OUT_62_P05 12
```

PWM_62 output configuration for P0.5 through PWM_COUT62.

16.21.2.25 PWM_OUT_62_P34

```
#define PWM_OUT_62_P34 13
```

PWM_62 output configuration for P3.4 through PWM_CC62.

16.21.2.26 PWM_OUT_62_P35

```
#define PWM_OUT_62_P35 14
```

PWM_62 output configuration for P3.5 through PWM_COUT62.

16.21.2.27 PWM_OUT_62_P46

```
#define PWM_OUT_62_P46 15
```

PWM_62 output configuration for P4.6 through PWM_CC62.

16.21.2.28 PWM_OUT_62_P47

```
#define PWM_OUT_62_P47 16
```

PWM_62 output configuration for P4.7 through PWM_COUT62.

16.21.2.29 PWM_OUT_63_P03

```
#define PWM_OUT_63_P03 17
```

PWM_63 output configuration for P0.3 through PWM_COUT63.

16.21.2.30 PWM_OUT_63_P37

```
#define PWM_OUT_63_P37 18
```

PWM_63 output configuration for P3.7 through PWM_COUT63.

16.21.2.31 PWM_OUT_63_P43

```
#define PWM_OUT_63_P43 19
```

PWM_63 output configuration for P4.3 through PWM_COUT63.

16.21.3 Typedef Documentation

16.21.3.1 hsk_pwm_channel

```
typedef ubyte hsk_pwm_channel
```

Type definition for PWM channels.

16.21.3.2 hsk_pwm_outChannel

```
typedef ubyte hsk_pwm_outChannel
```

Type definition for output channels.

16.21.3.3 hsk_pwm_port

```
typedef ubyte hsk_pwm_port
```

Type definition for ports.

16.21.4 Function Documentation

16.21.4.1 hsk_pwm_channel_set()

```
void hsk_pwm_channel_set (
    const hsk_pwm_channel channel,
    const uword max,
    const uword value )
```

Set the duty cycle for the given channel.

I.e. the active time frame slice of period can be set with max and value.

To set the duty cycle in percent specify a max of 100 and values from 0 to 100.

Parameters

<i>channel</i>	The PWM channel to set the duty cycle for, check the PWM_6x defines
<i>max</i>	Defines the scope value can move in
<i>value</i>	The current duty cycle value

16.21.4.2 hsk_pwm_disable()

```
void hsk_pwm_disable (
    void )
```

Deactivates the CCU6 to reduce power consumption.

16.21.4.3 hsk_pwm_enable()

```
void hsk_pwm_enable (
    void )
```

Turns on the CCU6.

Deactivates the power disable mode and sets the T12 and T13 Timer Run bits.

Precondition

All [hsk_pwm_init\(\)](#) calls have to be completed to call this

16.21.4.4 hsk_pwm_init()

```
void hsk_pwm_init (
    const hsk_pwm_channel channel,
    const ulong freq )
```

Sets up the the CCU6 timer frequencies that control the PWM cycle.

The channels PWM_60, PWM_61 and PWM_62 share the timer T12, thus initializing one of them, initializes them all. The channel PWM_63 has exclusive use of the timer T13 and can thus be used with its own operating frequency.

Frequencies up to $\sim 732.4\text{Hz}$ are always between 15 and 16 bits precision.

Frequencies above 48kHz offer less than 1/1000 precision. From there it is a linear function, i.e. 480kHz still offer 1/100 precision.

The freq value 0 will result in $\sim 0.02\text{Hz}$ ($48000000/2^{31}$).

The following formula results in the freq value that yields exactly the desired precision, this is useful to avoid precision loss by rounding:

$$freq(precision) = 480000000 * precision$$

E.g. 10 bit precision: $freq(1/2^{10}) = 468750$

Parameters

<i>channel</i>	The channel to change the frequency for
<i>freq</i>	The desired PWM cycle frequency in units of 0.1Hz

PWM Timings

The CCU6CLK can run at FCLK (48MHz) or PCLK (24MHz), configured in the CCUCCFG bit. This implementation always uses 48MHz.

The T12CLK can run any power of two between CCU6CLK and CCU6CLK/128, configured in the T12CLK bit field.

This value can additionally be multiplied with a prescaler of 1/256, activated with the T12PRE bit.

The same is true for the T13CLK.

Additionally the period is length for T12 and T13 can be configured to any 16 bit value. Assuming at least 1/1000 precision is desired that means the clock cycle can be shortened by any factor up to 2^6 (64).

The conclusion is that PWM frequencies between 48kHz and $\sim 0.02\text{Hz}$ can be configured. Very high values degrade the precision, e.g. 96kHz will only offer 1/500 precision. The freq value 0 will result in $\sim 0.02\text{Hz}$ ($48000000/2^{31}$).

16.21.4.5 hsk_pwm_outChannel_dir()

```
void hsk_pwm_outChannel_dir (
    hsk_pwm_outChannel channel,
    const bool up )
```

Set the direction of an output channel.

The channel value can be taken from any of the PWM_CCx/PWM_COUTx defines.

Parameters

<i>channel</i>	The IO channel to set the direction bit for
<i>up</i>	Set 1 to output a 1 during the cycle set with hsk_pwm_channel_set() , set 0 to output a 0 during the cycle set with hsk_pwm_channel_set()

16.21.4.6 hsk_pwm_port_close()

```
void hsk_pwm_port_close (
    const hsk_pwm_port port )
```

Close a PWM output port.

This configures the necessary port direction bits.

The port can be any one of the PWM_OUT_x_* defines.

Parameters

<i>port</i>	The output port to deactivate
-------------	-------------------------------

16.21.4.7 hsk_pwm_port_open()

```
void hsk_pwm_port_open (
    const hsk_pwm_port port )
```

Set up a PWM output port.

This configures the necessary port direction bits and activates the corresponding output channels.

The port can be any one of the PWM_OUT_x_* defines.

Precondition

This function should only be called after [hsk_pwm_enable\(\)](#), otherwise the output port will be driven (1) until PWM is enabled

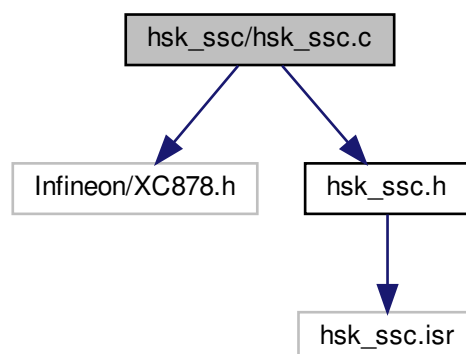
Parameters

<i>port</i>	The output port to activate
-------------	-----------------------------

16.22 hsk_ssc/hsk_ssc.c File Reference

HSK Synchronous Serial Interface implementation.

```
#include <Infineon/XC878.h>
#include "hsk_ssc.h"
Include dependency graph for hsk_ssc.c:
```



Macros

- #define [BIT_RMAP](#) 0
SYSCON0 Special Function Register Map Control bit.
- #define [BIT_EIR](#) 0
IRCON1 Error Interrupt Flag for SSC bit.
- #define [BIT_TIR](#) 1
IRCON1 Transmit Interrupt Flag for SSC bit.
- #define [BIT_RIR](#) 2
IRCON1 Receive Interrupt Flag for SSC bit.
- #define [BIT_SSC_DIS](#) 1
PMCON1 Disable Request bit.
- #define [BIT_MS](#) 6
SSC_CONH_P Master Select bit.
- #define [BIT_EIREN](#) 0
MODIEN Error Interrupt Enable Bit for SSC.
- #define [BIT_TIREN](#) 1
MODIEN Transmit Interrupt Enable Bit for SSC.
- #define [BIT_RIREN](#) 2
MODIEN Receive Interrupt Enable Bit for SSC.
- #define [BIT_MIS](#) 0
MODPISEL3 Master Mode Input Select bits.
- #define [BIT_SIS](#) 2
MODPISEL3 Slave Mode Input Select bits.
- #define [BIT_CIS](#) 4
MODPISEL3 Slave Mode Clock Input Select bits.
- #define [CNT_SEL](#) 2
Input Select bit count.
- #define [BIT_LB](#) 7
SSC_CONL Loop Back Control bit.
- #define [BIT_EN](#) 7
SSC_CONH_O Enable Bit.

Functions

- void [ISR_hsk_ssc](#) (void)
Transmit and receive interrupt.
- void [hsk_ssc_init](#) (const uword baud, const ubyte config, const bool mode)
The maximum baud rate in master mode is 12000000 bits/s, and 6000000 bits/s in slave mode.
- void [hsk_ssc_ports](#) (const ubyte ports)
Configure the I/O ports of the SSC unit.
- void [hsk_ssc_talk](#) (char *buffer, ubyte len)
Send and receive data.
- void [hsk_ssc_enable](#) ()
Turn the SSC module on.
- void [hsk_ssc_disable](#) ()
Turn the SSC module off.

Variables

- struct {
 - char * [rptr](#)
Pointer used for storing data read from the serial connection.
 - char * [wptr](#)
Pointer used to fetch data for writing on the serial connection.
 - ubyte [rcount](#)
Bytes left to read from the connection.
 - ubyte [wcount](#)
Bytes left to write on the connection.
- } [bufState](#)

Keeps the SSC communication state.

16.22.1 Detailed Description

HSK Synchronous Serial Interface implementation.

Note

The SFRs SSC_CONx_O and SSC_CONx_P refer to the same register address. The different suffixes signify the operation and programming modes in which the register exposes different bits.

Author

kami

16.22.2 Macro Definition Documentation

16.22.2.1 BIT_CIS

```
#define BIT_CIS 4
```

MODPISEL3 Slave Mode Clock Input Select bits.

16.22.2.2 BIT_EIR

```
#define BIT_EIR 0
```

IRCON1 Error Interrupt Flag for SSC bit.

16.22.2.3 BIT_EIREN

```
#define BIT_EIREN 0
```

MODIEN Error Interrupt Enable Bit for SSC.

16.22.2.4 BIT_EN

```
#define BIT_EN 7
```

SSC_CONH_O Enable Bit.

16.22.2.5 BIT_LB

```
#define BIT_LB 7
```

SSC_CONL Loop Back Control bit.

Half-duplex mode when set.

16.22.2.6 BIT_MIS

```
#define BIT_MIS 0
```

MODPISEL3 Master Mode Input Select bits.

16.22.2.7 BIT_MS

```
#define BIT_MS 6
```

SSC_CONH_P Master Select bit.

16.22.2.8 BIT_RIR

```
#define BIT_RIR 2
```

IRCON1 Receive Interrupt Flag for SSC bit.

16.22.2.9 BIT_RIREN

```
#define BIT_RIREN 2
```

MODIEN Receive Interrupt Enable Bit for SSC.

16.22.2.10 BIT_RMAP

```
#define BIT_RMAP 0
```

SYSCON0 Special Function Register Map Control bit.

16.22.2.11 BIT_SIS

```
#define BIT_SIS 2
```

MODPISEL3 Slave Mode Input Select bits.

16.22.2.12 BIT_SSC_DIS

```
#define BIT_SSC_DIS 1
```

PMCON1 Disable Request bit.

16.22.2.13 BIT_TIR

```
#define BIT_TIR 1
```

IRCON1 Transmit Interrupt Flag for SSC bit.

16.22.2.14 BIT_TIREN

```
#define BIT_TIREN 1
```

MODIEN Transmit Interrupt Enable Bit for SSC.

16.22.2.15 CNT_SEL

```
#define CNT_SEL 2
```

Input Select bit count.

16.22.3 Function Documentation

16.22.3.1 hsk_ssc_disable()

```
void hsk_ssc_disable ( )
```

Turn the SSC module off.

16.22.3.2 hsk_ssc_enable()

```
void hsk_ssc_enable ( )
```

Turn the SSC module on.

16.22.3.3 hsk_ssc_init()

```
void hsk_ssc_init (
    const uword baud,
    const ubyte config,
    const bool mode )
```

The maximum baud rate in master mode is 12000000 bits/s, and 6000000 bits/s in slave mode.

Calling this function turns the SSC off until [hsk_ssc_enable\(\)](#) is called.

Parameters

<i>baud</i>	The timer reload value for the baud rate generator, use SSC_BAUD to generate this value
<i>config</i>	The SSC configuration byte, use SSC_CONF to generate it
<i>mode</i>	Select master or slave operation

16.22.3.4 hsk_ssc_ports()

```
void hsk_ssc_ports (
    const ubyte ports )
```

Configure the I/O ports of the SSC unit.

Warning

Do not use when the SSC is enabled.

Parameters

<i>ports</i>	Selects an SSC I/O Ports I/O port configuration
--------------	---

16.22.3.5 hsk_ssc_talk()

```
void hsk_ssc_talk (
    char * buffer,
    ubyte len )
```

Send and receive data.

The buffer with the given length should contain the data to transceive and will be filled with the received data upon completion.

The provided buffer needs to reside in xdata memory, e.g. to create and use a string buffer the following should work:

```
char xdata buffer[] = "20 character buffer.";
...
hsk_ssc_talk(buffer, sizeof(buffer) - 1);
```

Note that char must not be const and that `sizeof(buffer) - 1` is used to prevent sending and overwriting the terminal 0 character. There may be cases where a terminal 0 character is desired.

Parameters

<i>buffer</i>	The rx/tx transmission buffer
<i>len</i>	The length of the buffer

16.22.3.6 ISR_hsk_ssc()

```
void ISR_hsk_ssc (
    void )
```

Transmit and receive interrupt.

16.22.4 Variable Documentation

16.22.4.1 bufState

```
bufState [static]
```

Keeps the SSC communication state.

16.22.4.2 rcount

```
ubyte rcount
```

Bytes left to read from the connection.

16.22.4.3 rptr

```
char* rptr
```

Pointer used for storing data read from the serial connection.

16.22.4.4 wcount

```
ubyte wcount
```

Bytes left to write on the connection.

16.22.4.5 wptr

```
char* wptr
```

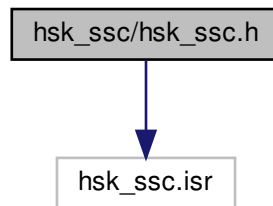
Pointer used to fetch data for writing on the serial connection.

16.23 hsk_ssc/hsk_ssc.h File Reference

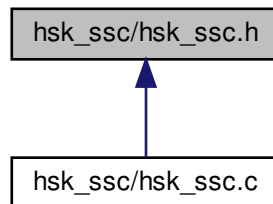
HSK Synchronous Serial Interface headers.

```
#include "hsk_ssc.isr"
```

Include dependency graph for hsk_ssc.h:



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



Macros

- `#define SSC_MRST_P05 1`
Master mode RX, slave mode TX port P0.5.
- `#define SSC_MRST_P14 0`
Master mode RX, slave mode TX port P1.4.
- `#define SSC_MRST_P15 2`
Master mode RX, slave mode TX port P1.5.
- `#define SSC_MTSR_P04 (1 << 2)`
Master mode TX, slave mode RX port P0.4.
- `#define SSC_MTSR_P13 (0 << 2)`
Master mode TX, slave mode RX port P1.3.
- `#define SSC_MTSR_P14 (2 << 2)`
Master mode TX, slave mode RX port P1.4.

- `#define SSC_SCLK_P03` (1 << 4)
Synchronous clock port P0.3.
- `#define SSC_SCLK_P12` (0 << 4)
Synchronous clock port P1.2.
- `#define SSC_SCLK_P13` (2 << 4)
Synchronous clock port P1.3.
- `#define SSC_MASTER` 1
Master mode, output shift clock on SCLK.
- `#define SSC_SLAVE` 0
Slave mode, receive shift clock on SCLK.
- `#define SSC_BAUD(bps)` (uword)(12000000ul / (bps) - 1)
Converts a baud rate value in bits/s into a baud rate value for the `hsk_ssc_init()` function.
- `#define SSC_CONF(width, heading, phase, polarity, duplex)` (((width) - 1) | ((heading) << 4) | ((phase) << 5) | ((polarity) << 6) | ((duplex) << 7))
Generates an SSC configuration byte.
- `#define hsk_ssc_busy()` ESSC
Returns whether the SSC is currently busy with data transmission.

Functions

- void `hsk_ssc_init` (const uword baud, const ubyte config, const bool mode)
The maximum baud rate in master mode is 12000000 bits/s, and 6000000 bits/s in slave mode.
- void `hsk_ssc_ports` (const ubyte ports)
Configure the I/O ports of the SSC unit.
- void `hsk_ssc_talk` (char *buffer, ubyte len)
Send and receive data.
- void `hsk_ssc_enable` ()
Turn the SSC module on.
- void `hsk_ssc_disable` ()
Turn the SSC module off.

16.23.1 Detailed Description

HSK Synchronous Serial Interface headers.

General purpose serial communication, setup in the following order:

- `hsk_ssc_init()`
- `hsk_ssc_ports()`
- `hsk_ssc_enable()`

Communication is established by the `hsk_ssc_talk()` function. Use `hsk_ssc_busy()` to detect whether a buffer was completely read and written.

Author

kami

16.23.2 Half Duplex Operation

For half duplex operation TX and RX pins need to be short circuited.

The TX pin is set up in open drain mode, i.e. an external pull-up resistor is required.

The TX pin needs to be manually configured before calling `hsk_ssc_talk()` in order to speak or listen on the bus. To listen the TX pin needs to be configured as an input pin, to speak on the bus as an output pin. For efficiency reasons this is not handled by this library (it would result in lots of runtime logic for what should be a single instruction).

Instead it is recommended to define macros in a central header. E.g. for the port configuration `SSC_MRST_P05` in slave mode the following code would work:

```
#define SSC_TX()      (P0_DIR |= 1 << 5)
#define SSC_RX()      (P0_DIR &= ~(1 << 5))
```

Syntactically it can be used like a regular function:

```
SSC_TX();
hsk_ssc_talk(buffer, sizeof(buffer) - 1);
```

16.23.3 Macro Definition Documentation

16.23.3.1 hsk_ssc_busy

```
#define hsk_ssc_busy( ) ESSC
```

Returns whether the SSC is currently busy with data transmission.

16.23.3.2 SSC_BAUD

```
#define SSC_BAUD(
    bps ) (uword)(12000000ul / (bps) - 1)
```

Converts a baud rate value in bits/s into a baud rate value for the `hsk_ssc_init()` function.

The distance between adjustable baud rates grows exponentially. Available baud rates in kBit progress like this:

{12000, 6000, 4000, 3000, 2400, 2000, ...}

Use the following formula to determine the baud rate that results from a desired value:

$$realBps(bps) = \frac{12000000}{\lfloor \frac{12000000}{bps} \rfloor}$$

Note

The maximum speed is 12 Mbit/s in master mode and 6 Mbit/s in slave mode.

Parameters

<i>bps</i>	The desired number in bits/s
------------	------------------------------

Returns

A timer reload value

16.23.3.3 SSC_CONF

```
#define SSC_CONF(
    width,
    heading,
    phase,
    polarity,
    duplex ) (((width) - 1) | ((heading) << 4) | ((phase) << 5) | ((polarity) << 6)
| ((duplex) << 7))
```

Generates an SSC configuration byte.

For details check the XC878 user manual section 12.3.5.1.

Parameters

<i>width</i>	The data with in bits, the available range is [2; 8]
<i>heading</i>	Use 0 for transmitting/receiving LSB first, 1 for MSB first
<i>phase</i>	Use 0 to shift on leading and latch on trailing edge, use 1 to shift on trailing and latch on leading edge
<i>polarity</i>	Use 0 for low idle clock, and 1 for high idle clock
<i>duplex</i>	Use 0 for full duplex mode and 1 for half duplex

16.23.3.4 SSC_MASTER

```
#define SSC_MASTER 1
```

Master mode, output shift clock on SCLK.

16.23.3.5 SSC_SLAVE

```
#define SSC_SLAVE 0
```

Slave mode, receive shift clock on SCLK.

16.23.4 Function Documentation**16.23.4.1 hsk_ssc_disable()**

```
void hsk_ssc_disable ( )
```

Turn the SSC module off.

16.23.4.2 hsk_ssc_enable()

```
void hsk_ssc_enable ( )
```

Turn the SSC module on.

16.23.4.3 hsk_ssc_init()

```
void hsk_ssc_init (
    const uword baud,
    const ubyte config,
    const bool mode )
```

The maximum baud rate in master mode is 12000000 bits/s, and 6000000 bits/s in slave mode.

Calling this function turns the SSC off until [hsk_ssc_enable\(\)](#) is called.

Parameters

<i>baud</i>	The timer reload value for the baud rate generator, use SSC_BAUD to generate this value
<i>config</i>	The SSC configuration byte, use SSC_CONF to generate it
<i>mode</i>	Select master or slave operation

16.23.4.4 hsk_ssc_ports()

```
void hsk_ssc_ports (
    const ubyte ports )
```

Configure the I/O ports of the SSC unit.

Warning

Do not use when the SSC is enabled.

Parameters

<i>ports</i>	Selects an SSC I/O Ports I/O port configuration
--------------	---

16.23.4.5 hsk_ssc_talk()

```
void hsk_ssc_talk (
    char * buffer,
    ubyte len )
```

Send and receive data.

The buffer with the given length should contain the data to transceive and will be filled with the received data upon completion.

The provided buffer needs to reside in xdata memory, e.g. to create and use a string buffer the following should work:

```
char xdata buffer[] = "20 character buffer.";
...
hsk_ssc_talk(buffer, sizeof(buffer) - 1);
```

Note that char must not be const and that $sizeof(buffer) - 1$ is used to prevent sending and overwriting the terminal 0 character. There may be cases where a terminal 0 character is desired.

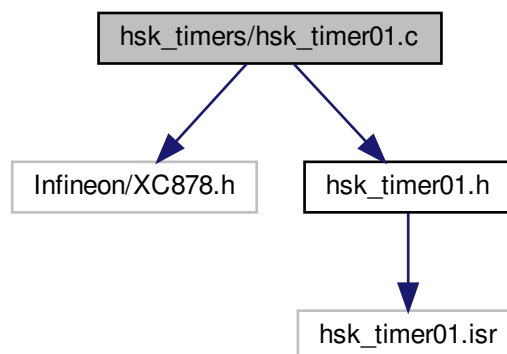
Parameters

<i>buffer</i>	The rx/tx transmission buffer
<i>len</i>	The length of the buffer

16.24 hsk_timers/hsk_timer01.c File Reference

HSK Timer 0/1 implementation.

```
#include <Infineon/XC878.h>
#include "hsk_timer01.h"
Include dependency graph for hsk_timer01.c:
```



Macros

- `#define BIT_ET0 1`
IEN0 Timer 0 Overflow Interrupt Enable bit.
- `#define BIT_ET1 3`
IEN0 Timer 1 Overflow Interrupt Enable bit.
- `#define BIT_TOM 0`
TMOD Timer 0 Mode select bits.

- `#define CNT_T0M 2`
T0M bit count.
- `#define BIT_T1M 4`
TMOD Timer 0 Mode select bits.
- `#define CNT_T1M 2`
T1M bit count.
- `#define BIT_RMAP 0`
SYSCON0 Special Function Register Map Control bit.

Functions

- `void ISR_hsk_timer0 (void)`
The ISR for timer 0.
- `void ISR_hsk_timer1 (void)`
The ISR for timer 1.
- `void hsk_timer01_setup (const ubyte id, const uword interval, const void(*const callback)(void))`
Setup timer 0 or 1 to tick at a given interval.
- `void hsk_timer0_setup (const uword interval, const void(*const callback)(void))`
Setup timer 0 to tick at a given interval.
- `void hsk_timer0_enable (void)`
Enables the timer 0 and its interrupt.
- `void hsk_timer0_disable (void)`
Disables timer 0 and its interrupt.
- `void hsk_timer1_setup (const uword interval, const void(*const callback)(void))`
Setup timer 1 to tick at a given interval.
- `void hsk_timer1_enable (void)`
Enables the timer 1 and its interrupt.
- `void hsk_timer1_disable (void)`
Disables timer 1 and its interrupt.

Variables

- `struct {`
 uword overflow
 The value to load into the timer upon overflow.
 void(* callback)(void)
 A callback function pointer used by the ISR.
} `timers [2]`

Struct representing runtime information for a timer.

16.24.1 Detailed Description

HSK Timer 0/1 implementation.

This simple library implements access to the timers T0 and T1, as 16 bit timers to use as a ticking source.

Author

kami

16.24.2 Macro Definition Documentation

16.24.2.1 BIT_ET0

```
#define BIT_ET0 1
```

IEN0 Timer 0 Overflow Interrupt Enable bit.

16.24.2.2 BIT_ET1

```
#define BIT_ET1 3
```

IEN0 Timer 1 Overflow Interrupt Enable bit.

16.24.2.3 BIT_RMAP

```
#define BIT_RMAP 0
```

SYSCON0 Special Function Register Map Control bit.

16.24.2.4 BIT_T0M

```
#define BIT_T0M 0
```

TMOD Timer 0 Mode select bits.

16.24.2.5 BIT_T1M

```
#define BIT_T1M 4
```

TMOD Timer 0 Mode select bits.

16.24.2.6 CNT_T0M

```
#define CNT_T0M 2
```

T0M bit count.

16.24.2.7 CNT_T1M

```
#define CNT_T1M 2
```

T1M bit count.

16.24.3 Function Documentation

16.24.3.1 hsk_timer01_setup()

```
void hsk_timer01_setup (  
    const ubyte id,  
    const uword interval,  
    const void(*) (void) callback ) [private]
```

Setup timer 0 or 1 to tick at a given interval.

The callback function will be called by the interrupt once the interrupt has been enabled. Note that the callback function is entered with the current page unknown.

This works on the assumption, that PCLK is set to 24MHz.

Parameters

<i>id</i>	Timer 0 or 1.
<i>interval</i>	The ticking interval in μ s, don't go beyond 5461.
<i>callback</i>	A function pointer to a callback function.

The timer ticks with $PCLK / 2$, which means 12 timer ticks per μ s.

16.24.3.2 hsk_timer0_disable()

```
void hsk_timer0_disable (
    void )
```

Disables timer 0 and its interrupt.

16.24.3.3 hsk_timer0_enable()

```
void hsk_timer0_enable (
    void )
```

Enables the timer 0 and its interrupt.

16.24.3.4 hsk_timer0_setup()

```
void hsk_timer0_setup (
    const uword interval,
    const void(*) (void) callback )
```

Setup timer 0 to tick at a given interval.

The callback function will be called by the interrupt once the interrupt has been enabled. Note that the callback function is entered with the current page unknown.

This works on the assumption, that PCLK is set to 24MHz.

Parameters

<i>interval</i>	The ticking interval in μ s, don't go beyond 5461.
<i>callback</i>	A function pointer to a callback function.

Here is the call graph for this function:



16.24.3.5 hsk_timer1_disable()

```
void hsk_timer1_disable (
    void )
```

Disables timer 1 and its interrupt.

16.24.3.6 hsk_timer1_enable()

```
void hsk_timer1_enable (
    void )
```

Enables the timer 1 and its interrupt.

16.24.3.7 hsk_timer1_setup()

```
void hsk_timer1_setup (
    const uword interval,
    const void(*) (void) callback )
```

Setup timer 1 to tick at a given interval.

The callback function will be called by the interrupt once the interrupt has been enabled. Note that the callback function is entered with the current page unknown.

This works on the assumption, that PCLK is set to 24MHz.

Parameters

<i>interval</i>	The ticking interval in μ s, don't go beyond 5461.
<i>callback</i>	A function pointer to a callback function.

Here is the call graph for this function:



16.24.3.8 ISR_hsk_timer0()

```
void ISR_hsk_timer0 (  
    void ) [private]
```

The ISR for timer 0.

Sets up the timer 0 count registers and calls the callback function.

16.24.3.9 ISR_hsk_timer1()

```
void ISR_hsk_timer1 (  
    void ) [private]
```

The ISR for timer 1.

Sets up the timer 1 count registers and calls the callback function.

16.24.4 Variable Documentation

16.24.4.1 callback

```
void( * callback) (void)
```

A callback function pointer used by the ISR.

16.24.4.2 overflow

```
uword overflow
```

The value to load into the timer upon overflow.

16.24.4.3 timers

```
timers [static]
```

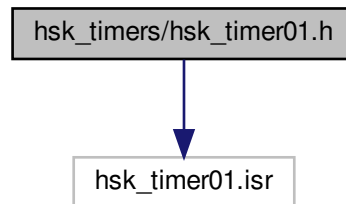
Struct representing runtime information for a timer.

16.25 hsk_timers/hsk_timer01.h File Reference

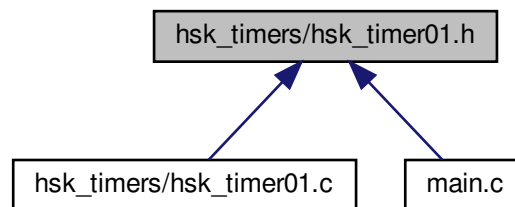
HSK Timer 0/1 headers.

```
#include "hsk_timer01.isr"
```

Include dependency graph for hsk_timer01.h:



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



Functions

- void [hsk_timer0_setup](#) (const uword interval, const void(*const [callback](#))(void))
Setup timer 0 to tick at a given interval.
- void [hsk_timer0_enable](#) (void)
Enables the timer 0 and its interrupt.
- void [hsk_timer0_disable](#) (void)
Disables timer 0 and its interrupt.
- void [hsk_timer1_setup](#) (const uword interval, const void(*const [callback](#))(void))
Setup timer 1 to tick at a given interval.
- void [hsk_timer1_enable](#) (void)
Enables the timer 1 and its interrupt.
- void [hsk_timer1_disable](#) (void)
Disables timer 1 and its interrupt.

16.25.1 Detailed Description

HSK Timer 0/1 headers.

Provides access to the timers 0 and 1. Each timer can be provided with a callback function that will be called by the timers ISR.

Author

kami

16.25.2 Function Documentation

16.25.2.1 hsk_timer0_disable()

```
void hsk_timer0_disable (
    void )
```

Disables timer 0 and its interrupt.

16.25.2.2 hsk_timer0_enable()

```
void hsk_timer0_enable (
    void )
```

Enables the timer 0 and its interrupt.

16.25.2.3 hsk_timer0_setup()

```
void hsk_timer0_setup (
    const uword interval,
    const void(*) (void) callback )
```

Setup timer 0 to tick at a given interval.

The callback function will be called by the interrupt once the interrupt has been enabled. Note that the callback function is entered with the current page unknown.

This works on the assumption, that PCLK is set to 24MHz.

Parameters

<i>interval</i>	The ticking interval in μ s, don't go beyond 5461.
<i>callback</i>	A function pointer to a callback function.

Here is the call graph for this function:



16.25.2.4 hsk_timer1_disable()

```
void hsk_timer1_disable (
    void )
```

Disables timer 1 and its interrupt.

16.25.2.5 hsk_timer1_enable()

```
void hsk_timer1_enable (
    void )
```

Enables the timer 1 and its interrupt.

16.25.2.6 hsk_timer1_setup()

```
void hsk_timer1_setup (
    const uword interval,
    const void(*) (void) callback )
```

Setup timer 1 to tick at a given interval.

The callback function will be called by the interrupt once the interrupt has been enabled. Note that the callback function is entered with the current page unknown.

This works on the assumption, that PCLK is set to 24MHz.

Parameters

<i>interval</i>	The ticking interval in μ s, don't go beyond 5461.
<i>callback</i>	A function pointer to a callback function.

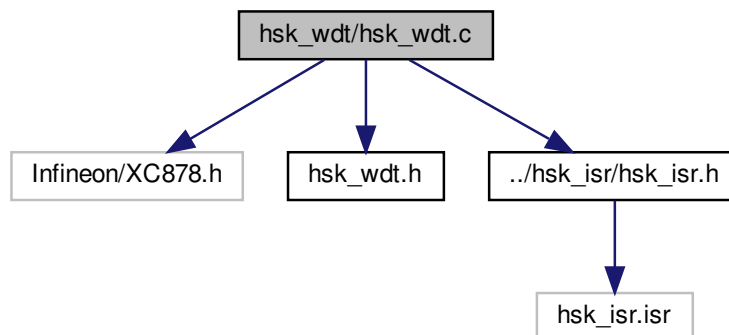
Here is the call graph for this function:



16.26 hsk_wdt/hsk_wdt.c File Reference

HSK Watchdog Timer implementation.

```
#include <Infineon/XC878.h>
#include "hsk_wdt.h"
#include "../hsk_isr/hsk_isr.h"
Include dependency graph for hsk_wdt.c:
```



Macros

- `#define BIT_WDTIN 0`
WDTCN Watchdog Timer Input Frequency Selection bit.
- `#define BIT_WDTEN 2`
WDTCN WDT Enable bit.
- `#define BIT_WDTRS 1`
WDTCN WDT Refresh Start bit.

Functions

- void [hsk_wdt_init](#) (const uword window)
Sets up the watchdog timer.
- void [hsk_wdt_enable](#) (void)
Activates the Watchdog Timer.
- void [hsk_wdt_disable](#) (void)
Disables the Watchdog Timer.
- void [hsk_wdt_service](#) (void)
Resets the watchdog timer.

16.26.1 Detailed Description

HSK Watchdog Timer implementation.

The WDT is a 16bit counter that counts up, upon overflow a reset is initiated. When the timer is serviced the higher byte is loaded from the WDTREL SFR.

Author

kami

16.26.2 Watchdog Timer Registers

All registers are in the mapped register area, i.e. RMAP=1 must be set to access them.

16.26.3 Macro Definition Documentation

16.26.3.1 BIT_WDTEN

```
#define BIT_WDTEN 2
```

WDTCON WDT Enable bit.

This bit is protected.

16.26.3.2 BIT_WDTIN

```
#define BIT_WDTIN 0
```

WDTCON Watchdog Timer Input Frequency Selection bit.

Used to select PCLK/128 instead of PCLK/2.

16.26.3.3 BIT_WDTRS

```
#define BIT_WDTRS 1
```

WDTCON WDT Refresh Start bit.

16.26.4 Function Documentation

16.26.4.1 hsk_wdt_disable()

```
void hsk_wdt_disable (
    void )
```

Disables the Watchdog Timer.

16.26.4.2 hsk_wdt_enable()

```
void hsk_wdt_enable (
    void )
```

Activates the Watchdog Timer.

16.26.4.3 hsk_wdt_init()

```
void hsk_wdt_init (
    const uword window )
```

Sets up the watchdog timer.

The window time specifies the time available to call [hsk_wdt_service\(\)](#) before a reset is triggered. Possible times range from 21.3μs to 350ms.

The window time is rounded up to the next higher possible value. Exceeding the value range causes an overflow that results in shorter window times.

Parameters

<i>window</i>	The time window in multiples of 10μs
---------------	--------------------------------------

The WDT runs at PCLK/2 or PCLK/128, i.e. the WDT low byte WDTL overflow occurs either every 21.333μs or every 1365.333ms.

One time unit (10μs) equals 120 PCLK/2 clock ticks. One PCLK/128 time unit (640μs) equals 120 PCLK/128 clock ticks.

16.26.4.4 hsk_wdt_service()

```
void hsk_wdt_service (
    void )
```

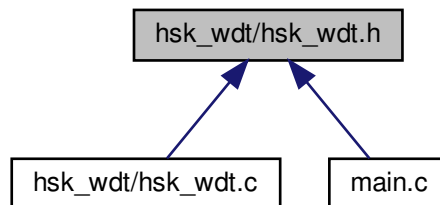
Resets the watchdog timer.

This function needs to be called to prevent the WDT from resetting the device.

16.27 hsk_wdt/hsk_wdt.h File Reference

HSK Watchdog Timer headers.

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



Functions

- void [hsk_wdt_init](#) (const uword window)
Sets up the watchdog timer.
- void [hsk_wdt_enable](#) (void)
Activates the Watchdog Timer.
- void [hsk_wdt_disable](#) (void)
Disables the Watchdog Timer.
- void [hsk_wdt_service](#) (void)
Resets the watchdog timer.

16.27.1 Detailed Description

HSK Watchdog Timer headers.

Provides access to the Watchdog Timer (WDT) of the XC878.

Depending on the configured window time the μ C reset is delayed for 1.024ms (window < 5460 μ s) or 65.536ms (window \geq 5460 μ s).

This time can be used by assigning a callback function to [hsk_isr14](#) member [hsk_isr14_callback::NMIWDT](#) and setting the NMICON.NMIWDT bit.

Warning

The WDT should be set up at the end of the boot procedure. Setting the WDT up at the beginning of the boot process can trigger all kinds of erratic behaviour like reset races or a complete lockup.

Author

kami

16.27.2 Hazards

The WDT has proven a useful tool in hazardous EMI conditions. Severe EMI may freeze the μ C without causing a proper reboot. In most cases the WDT can mitigate this issue by reactivating the system.

However the WDT is trigger happy. A series of refresh time interval measurements shows that the WDT resets the μ C long before the end of its interval shortly after boot. The best mitigation is to refresh the WDT with the [hsk_wdt_service\(\)](#) function unconditionally. Instead of using fixed timings (e.g. for 20ms watchdog time a 5ms refresh interval should have been quite safe).

That however does not solve the problem with NMIs. Any non-maskable interrupt may cause the WDT to reset the μ C. This means it is incompatible to the hsk_flash library, which requires precise timings with only a couple of μ s tolerance. To meet this requirement the library uses the Flash Timer of the XC878, which triggers NMIs.

16.27.3 Function Documentation

16.27.3.1 hsk_wdt_disable()

```
void hsk_wdt_disable (
    void )
```

Disables the Watchdog Timer.

16.27.3.2 hsk_wdt_enable()

```
void hsk_wdt_enable (
    void )
```

Activates the Watchdog Timer.

16.27.3.3 hsk_wdt_init()

```
void hsk_wdt_init (
    const uword window )
```

Sets up the watchdog timer.

The window time specifies the time available to call [hsk_wdt_service\(\)](#) before a reset is triggered. Possible times range from 21.3 μ s to 350ms.

The window time is rounded up to the next higher possible value. Exceeding the value range causes an overflow that results in shorter window times.

Parameters

<i>window</i>	The time window in multiples of 10 μ s
---------------	--

The WDT runs at PCLK/2 or PCLK/128, i.e. the WDT low byte WDTL overflow occurs either every 21.333 μ s or every 1365.333ms.

One time unit (10µs) equals 120 PCLK/2 clock ticks. One PCLK/128 time unit (640µs) equals 120 PCLK/128 clock ticks.

16.27.3.4 hsk_wdt_service()

```
void hsk_wdt_service (
    void )
```

Resets the watchdog timer.

This function needs to be called to prevent the WDT from resetting the device.

16.28 main.c File Reference

Simple test file that is not linked into the library.

```
#include <Infineon/XC878.h>
#include "config.h"
#include "hsk_boot/hsk_boot.h"
#include "hsk_timers/hsk_timer01.h"
#include "hsk_can/hsk_can.h"
#include "hsk_icm7228/hsk_icm7228.h"
#include "hsk_adc/hsk_adc.h"
#include "hsk_pwm/hsk_pwm.h"
#include "hsk_pwc/hsk_pwc.h"
#include "hsk_flash/hsk_flash.h"
#include "hsk_wdt/hsk_wdt.h"
#include "hsk_io/hsk_io.h"
```

Include dependency graph for main.c:



Data Structures

- struct [hsk_flash_struct](#)

This struct is a template for data that can be written to the D-Flash.

Macros

- `#define PERSIST_VERSION 1`

The version of the persist struct.

Functions

- void `p1_init` (void)
Set up buffer and ports for display driver at I/O port P1 .
- void `p1_refresh` (void)
Refresh displays at I/O port P1 with the buffered data.
- void `p1_writeString` (char const *const str, ubyte const pos, ubyte const len)
Write an ASCII encoded string into `p1_buffer`.
- void `p1_writeDec` (uword const value, char const power, ubyte const pos, ubyte const len)
Write a decimal number into `p1_buffer`.
- void `p1_writeHex` (uword const value, char const power, ubyte const pos, ubyte const len)
Write a hexadecimal number into `p1_buffer`.
- void `p1_illuminate` (ubyte const segments, ubyte const pos, ubyte const len)
Illuminate a number of segments in `p1_buffer`.
- void `main` (void)
Call init functions and invoke the run routine.
- void `init` (void)
Initialize ports, timers and ISRs.
- void `run` (void)
The main test code body.
- void `tick0` (void)
A ticking function called back by the timer T0 ISR.

Variables

- ubyte `p1_buffer` [8]
Buffer for display driver at I/O port P1 .
- volatile struct `hsk_flash_struct` `persist`
This structure is used to persist data between resets.
- volatile uword `tick0_count_250` = 0
A counter used to detecting that 250ms have passed.
- volatile ubyte `tick0_count_20` = 10
A counter used to detecting that 20ms have passed.
- volatile uword `adc7`
The storage variable for the potentiometer on the eval board.

16.28.1 Detailed Description

Simple test file that is not linked into the library.

This file is normally rigged to run on the XC800 Starter Kit eval board and used for testing whatever code is currently under development.

Author

kami

16.28.2 Macro Definition Documentation

16.28.2.1 PERSIST_VERSION

```
#define PERSIST_VERSION 1
```

The version of the persist struct.

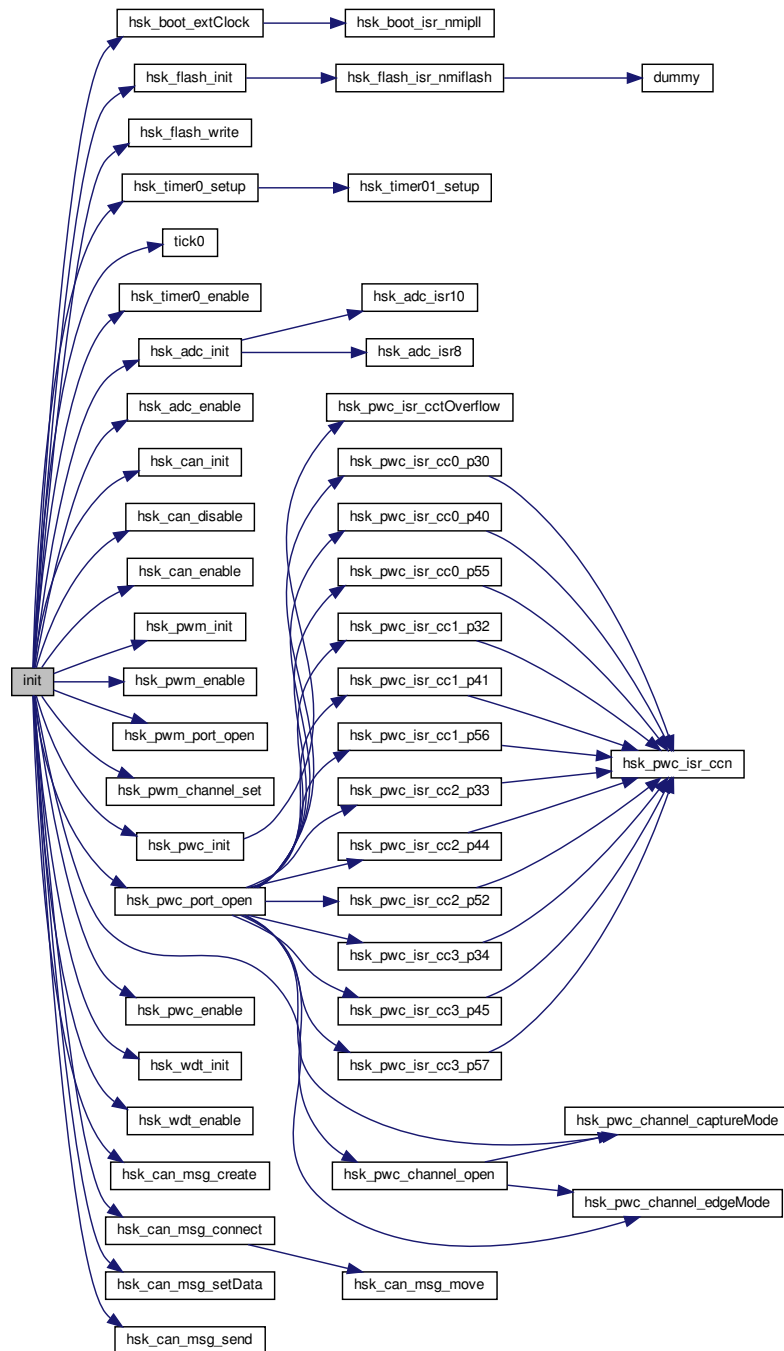
16.28.3 Function Documentation

16.28.3.1 init()

```
void init (
    void )
```

Initialize ports, timers and ISRs.

Here is the call graph for this function:

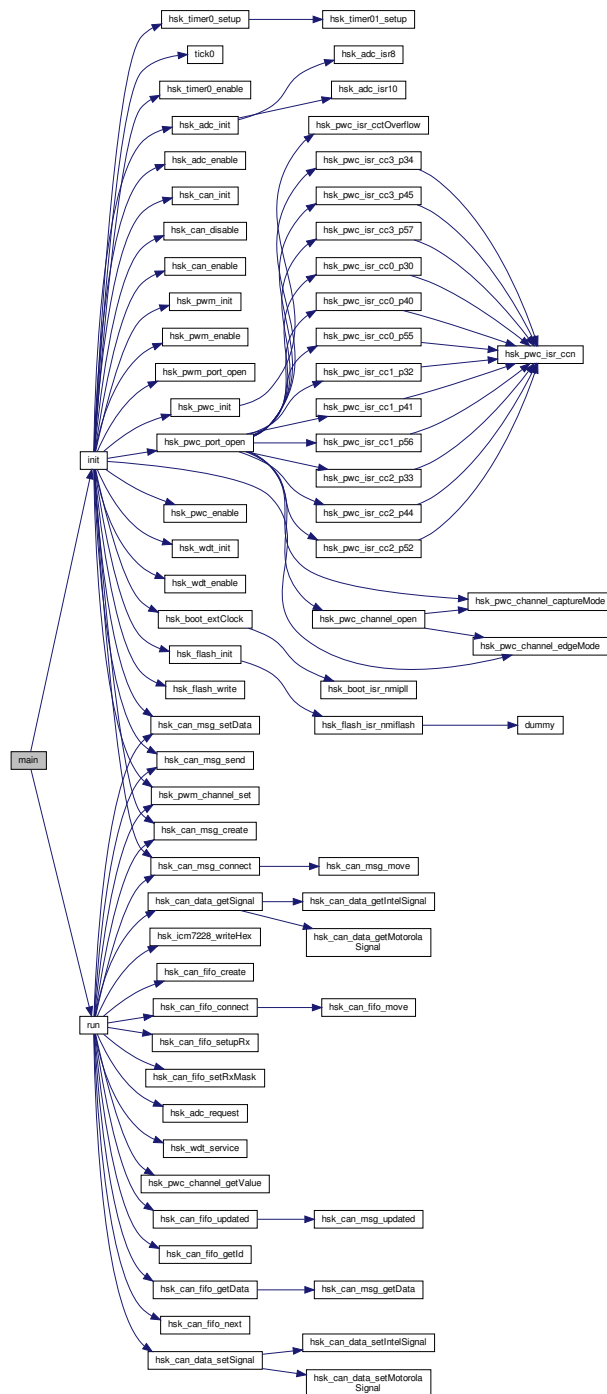


16.28.3.2 main()

```
void main (
    void )
```

Call init functions and invoke the run routine.

Here is the call graph for this function:



16.28.3.3 p1_illuminate()

```

void p1_illuminate (
    ubyte const segments,
    ubyte const pos,
    ubyte const len ) [inline]

```

Illuminate a number of segments in [p1_buffer](#).

Parameters

<i>segments</i>	The number of segments to illuminate
<i>pos</i>	The target position in the buffer
<i>len</i>	The number of digits available to encode the number

See also

[ICM7228_FACTORY](#)
[hsk_icm7228_illuminate](#)

16.28.3.4 p1_init()

```
void p1_init (
    void )
```

Set up buffer and ports for display driver at I/O port P1 .

See also

[ICM7228_FACTORY](#)

16.28.3.5 p1_refresh()

```
void p1_refresh (
    void )
```

Refresh displays at I/O port P1 with the buffered data.

See also

[ICM7228_FACTORY](#)

16.28.3.6 p1_writeDec()

```
void p1_writeDec (
    uword const value,
    char const power,
    ubyte const pos,
    ubyte const len ) [inline]
```

Write a decimal number into [p1_buffer](#).

Parameters

<i>value</i>	The number to encode
<i>power</i>	The 10 base power of the number to encode
<i>pos</i>	The target position in the buffer
<i>len</i>	The number of digits available to encode the number

See also

[ICM7228_FACTORY](#)
[hsk_icm7228_writeDec](#)

16.28.3.7 p1_writeHex()

```
void p1_writeHex (
    uword const value,
    char const power,
    ubyte const pos,
    ubyte const len ) [inline]
```

Write a hexadecimal number into [p1_buffer](#).

Parameters

<i>value</i>	The number to encode
<i>power</i>	The 16 base power of the number to encode
<i>pos</i>	The target position in the buffer
<i>len</i>	The number of digits available to encode the number

See also

[ICM7228_FACTORY](#)
[hsk_icm7228_writeHex](#)

16.28.3.8 p1_writeString()

```
void p1_writeString (
    char const *const str,
    ubyte const pos,
    ubyte const len ) [inline]
```

Write an ASCII encoded string into [p1_buffer](#).

Parameters

<i>str</i>	The buffer to read the ASCII string from
<i>pos</i>	The position in the buffer to write the encoded string to
<i>len</i>	The target length of the encoded string

See also

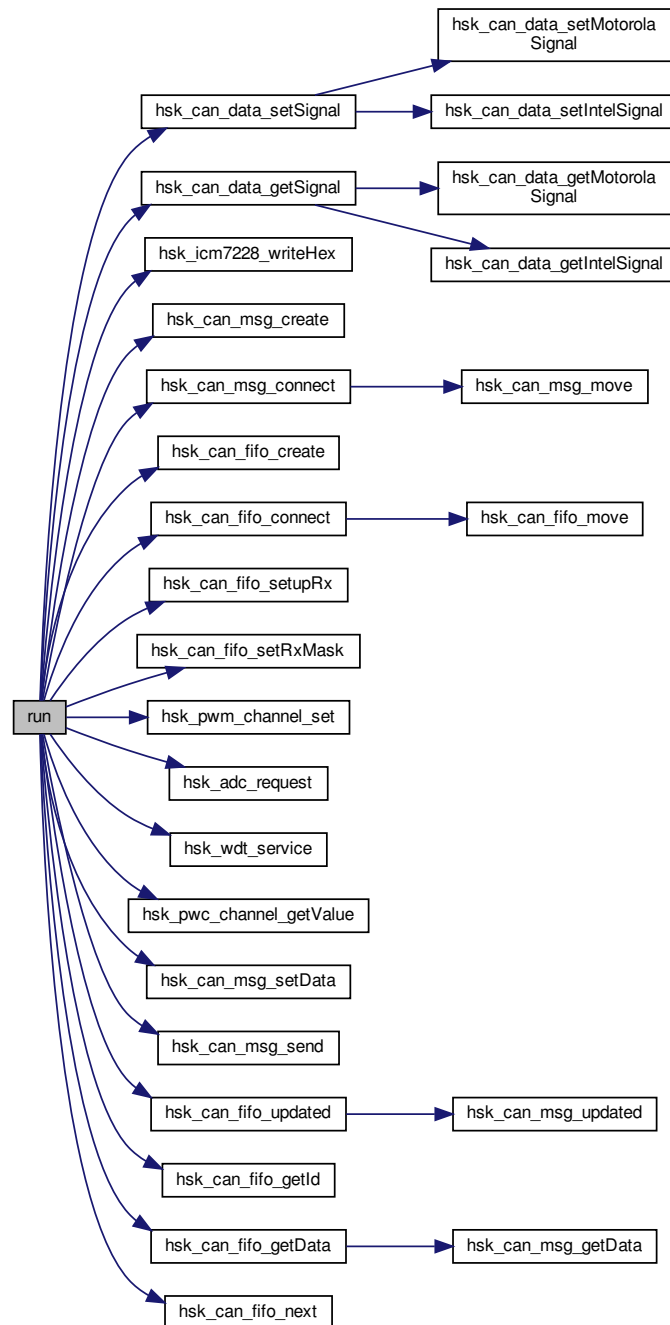
[ICM7228_FACTORY](#)
[hsk_icm7228_writeString](#)

16.28.3.9 run()

```
void run (
    void )
```

The main test code body.

Here is the call graph for this function:



16.28.3.10 tick0()

```
void tick0 (
    void )
```

A ticking function called back by the timer T0 ISR.

16.28.4 Variable Documentation

16.28.4.1 adc7

```
volatile uword adc7
```

The storage variable for the potentiometer on the eval board.

16.28.4.2 p1_buffer

```
ubyte p1_buffer[8]
```

Buffer for display driver at I/O port P1 .

See also

[ICM7228_FACTORY](#)

16.28.4.3 persist

```
persist
```

This structure is used to persist data between resets.

16.28.4.4 tick0_count_20

```
volatile ubyte tick0_count_20 = 10
```

A counter used to detecting that 20ms have passed.

16.28.4.5 tick0_count_250

```
volatile uword tick0_count_250 = 0
```

A counter used to detecting that 250ms have passed.

Index

`_sdcc_external_startup`
 `hsk_boot.c`, [113](#)

`ADC_CHANNELS`
 `hsk_adc.c`, [93](#)
`ADC_CLK_12MHz`
 `hsk_adc.c`, [93](#)
`ADC_CLK_6MHz`
 `hsk_adc.c`, [93](#)
`ADC_CLK_750kHz`
 `hsk_adc.c`, [94](#)
`ADC_CLK_8MHz`
 `hsk_adc.c`, [94](#)
`ADC_QUEUE`
 `hsk_adc.c`, [94](#)
`ADC_RESOLUTION_10`
 `hsk_adc.h`, [103](#)
`ADC_RESOLUTION_8`
 `hsk_adc.h`, [103](#)
`ADCSR0`
 `hsk_isr6_callback`, [82](#)
`ADCSR1`
 `hsk_isr6_callback`, [82](#)
`ADDR_DFLASH`
 `hsk_flash.c`, [191](#)
`ADDR_PFLASH`
 `hsk_flash.c`, [191](#)
`ADDR_ROM`
 `hsk_flash.c`, [191](#)
`ADDR_XRAM`
 `hsk_flash.c`, [191](#)
`AUAD_DEC1`
 `hsk_can.c`, [125](#)
`AUAD_INC1`
 `hsk_can.c`, [125](#)
`AUAD_INC8`
 `hsk_can.c`, [126](#)
`AUAD_OFF`
 `hsk_can.c`, [126](#)
`adc7`
 `main.c`, [313](#)
`averageOver`
 `hsk_pwc.c`, [246](#)

`BIT_ADC_DIS`
 `hsk_adc.c`, [94](#)
`BIT_ADCSR0`
 `hsk_isr.c`, [220](#)
`BIT_ADCSR1`
 `hsk_isr.c`, [220](#)

`BIT_ALIE`
 `hsk_can.c`, [126](#)
`BIT_ANON`
 `hsk_adc.c`, [94](#)
`BIT_ASEN_PARALLEL`
 `hsk_adc.c`, [94](#)
`BIT_ASEN_SEQUENTIAL`
 `hsk_adc.c`, [94](#)
`BIT_AUAD`
 `hsk_can.c`, [126](#)
`BIT_AM`
 `hsk_can.c`, [126](#)
`BIT_BRP`
 `hsk_can.c`, [126](#)
`BIT_BSY`
 `hsk_can.c`, [126](#)
`BIT_BUSY`
 `hsk_can.c`, [126](#)
`BIT_CALM`
 `hsk_can.c`, [127](#)
`BIT_CAN_DIS`
 `hsk_can.c`, [127](#)
`BIT_CANDIS`
 `hsk_can.c`, [127](#)
`BIT_CANSRC0`
 `hsk_isr.c`, [220](#)
`BIT_CANSRC1`
 `hsk_isr.c`, [220](#)
`BIT_CANSRC2`
 `hsk_isr.c`, [220](#)
`BIT_CANSRC3`
 `hsk_isr.c`, [220](#)
`BIT_CCM0`
 `hsk_pwc.c`, [234](#)
`BIT_CCTBx`
 `hsk_pwc.c`, [234](#)
`BIT_CCTOVEN`
 `hsk_pwc.c`, [234](#)
`BIT_CCTOVF`
 `hsk_isr.c`, [220](#)
 `hsk_pwc.c`, [234](#)
`BIT_CCTPRE`
 `hsk_pwc.c`, [234](#)
`BIT_CCTST`
 `hsk_pwc.c`, [235](#)
`BIT_CCU_DIS`
 `hsk_pwm.c`, [262](#)
`BIT_CCUCFG`
 `hsk_pwm.c`, [262](#)

BIT_CCE
 hsk_can.c, 127
 BIT_CHNR
 hsk_adc.c, 94
 BIT_CIS
 hsk_ssc.c, 280
 BIT_CTC
 hsk_adc.c, 95
 BIT_DATA
 hsk_can.c, 127
 BIT_DIV8
 hsk_can.c, 127
 BIT_DIR
 hsk_can.c, 127
 BIT_DLC
 hsk_can.c, 128
 BIT_DW
 hsk_adc.c, 95
 BIT_ECT130
 hsk_pwm.c, 263
 BIT_EEABORT
 hsk_flash.c, 192
 BIT_EEBSY
 hsk_flash.c, 192
 BIT_EIREN
 hsk_ssc.c, 280
 BIT_EIR
 hsk_ssc.c, 280
 BIT_EMPTY
 hsk_adc.c, 95
 BIT_ENGT
 hsk_adc.c, 95
 BIT_EOFSYN
 hsk_isr.c, 221
 BIT_EORDRES
 hsk_boot.c, 110
 BIT_EOC
 hsk_isr.c, 221
 BIT_ERASE
 hsk_flash.c, 192
 BIT_ERRSYN
 hsk_isr.c, 221
 BIT_ERR
 hsk_can.c, 128
 BIT_ET0
 hsk_timer01.c, 292
 BIT_ET1
 hsk_timer01.c, 292
 BIT_EXF2
 hsk_isr.c, 221
 BIT_EXINT0
 hsk_ex.c, 178
 BIT_EXINT1
 hsk_ex.c, 178
 BIT_EXINT2
 hsk_ex.c, 178
 hsk_isr.c, 221
 BIT_EXINT3
 hsk_ex.c, 178
 hsk_isr.c, 221
 BIT_EXINT4
 hsk_ex.c, 178
 hsk_isr.c, 222
 BIT_EXINT5
 hsk_ex.c, 179
 hsk_isr.c, 222
 BIT_EXINT6
 hsk_ex.c, 179
 hsk_isr.c, 222
 BIT_EXTOSCR
 hsk_boot.c, 110
 BIT_EN
 hsk_ssc.c, 280
 BIT_FCCFG
 hsk_can.c, 128
 BIT_FILL
 hsk_adc.c, 95
 BIT_FTEN
 hsk_flash.c, 192
 BIT_IDEXT
 hsk_can.c, 128
 BIT_IDSTD
 hsk_can.c, 128
 BIT_IDE
 hsk_can.c, 128
 BIT_IERR
 hsk_isr.c, 222
 BIT_IEN
 hsk_adc.c, 95
 BIT_IMODE
 hsk_adc.c, 95
 hsk_ex.c, 179
 hsk_pwc.c, 235
 BIT_INIT
 hsk_can.c, 128
 BIT_IRDY
 hsk_isr.c, 222
 BIT_LECIE
 hsk_can.c, 129
 BIT_LIST
 hsk_can.c, 129
 BIT_LB
 hsk_ssc.c, 281
 BIT_MAS1
 hsk_flash.c, 192
 BIT_MIDE
 hsk_can.c, 129
 BIT_MIS
 hsk_ssc.c, 281
 BIT_MMC
 hsk_can.c, 129
 BIT_MODE
 hsk_flash.c, 192
 BIT_MSGVAL
 hsk_can.c, 129
 BIT_MXB19

- hsk_boot.c, [110](#)
- BIT_MXB
 - hsk_boot.c, [110](#)
- BIT_MXM
 - hsk_boot.c, [110](#)
- BIT_MS
 - hsk_ssc.c, [281](#)
- BIT_NDIVH
 - hsk_boot.c, [110](#)
- BIT_NDIVL
 - hsk_boot.c, [110](#)
- BIT_NDOV
 - hsk_isr.c, [222](#)
- BIT_NEWDAT
 - hsk_can.c, [129](#)
- BIT_NMIECC
 - hsk_isr.c, [223](#)
- BIT_NMIFLASH
 - hsk_flash.c, [193](#)
 - hsk_isr.c, [223](#)
- BIT_NMIPLL
 - hsk_boot.c, [111](#)
 - hsk_isr.c, [223](#)
- BIT_NMIVDDP
 - hsk_isr.c, [223](#)
- BIT_NMIWDT
 - hsk_isr.c, [223](#)
- BIT_NVSTR
 - hsk_flash.c, [193](#)
- BIT_OFVAL
 - hsk_flash.c, [193](#)
- BIT_OSCSS
 - hsk_boot.c, [111](#)
- BIT_PDIV
 - hsk_boot.c, [111](#)
- BIT_PLL_LOCK
 - hsk_boot.c, [111](#)
- BIT_PLLBYP
 - hsk_boot.c, [111](#)
- BIT_PLLPD
 - hsk_boot.c, [111](#)
- BIT_PLLRDRES
 - hsk_boot.c, [112](#)
- BIT_PLLR
 - hsk_boot.c, [112](#)
- BIT_PROG
 - hsk_flash.c, [193](#)
- BIT_PRI
 - hsk_can.c, [129](#)
- BIT_PSL63
 - hsk_pwm.c, [263](#)
- BIT_PSL
 - hsk_pwm.c, [263](#)
- BIT_RBUSY
 - hsk_can.c, [129](#)
- BIT_REQCHNR
 - hsk_adc.c, [95](#)
- BIT_RESULT
 - hsk_adc.c, [96](#)
- BIT_RIREN
 - hsk_ssc.c, [281](#)
- BIT_RIR
 - hsk_ssc.c, [281](#)
- BIT_RMAP
 - hsk_isr.c, [223](#)
 - hsk_ssc.c, [281](#)
 - hsk_timer01.c, [292](#)
- BIT_RWEN
 - hsk_can.c, [130](#)
- BIT_RXEN
 - hsk_can.c, [130](#)
- BIT_RXPND
 - hsk_can.c, [130](#)
- BIT_RXSEL
 - hsk_can.c, [130](#)
- BIT_RXUPD
 - hsk_can.c, [130](#)
- BIT_RI
 - hsk_isr.c, [223](#)
- BIT_SIS
 - hsk_ssc.c, [281](#)
- BIT_SJW
 - hsk_can.c, [130](#)
- BIT_SSC_DIS
 - hsk_ssc.c, [282](#)
- BIT_T0M
 - hsk_timer01.c, [292](#)
- BIT_T1M
 - hsk_timer01.c, [292](#)
- BIT_T2CCFG
 - hsk_pwc.c, [235](#)
- BIT_T2CCU_DIS
 - hsk_pwc.c, [235](#)
- BIT_TF2
 - hsk_isr.c, [224](#)
- BIT_TIMSYN
 - hsk_pwc.c, [235](#)
- BIT_TIREN
 - hsk_ssc.c, [282](#)
- BIT_TIR
 - hsk_ssc.c, [282](#)
- BIT_TRIE
 - hsk_can.c, [130](#)
- BIT_TSEG1
 - hsk_can.c, [131](#)
- BIT_TSEG2
 - hsk_can.c, [131](#)
- BIT_TXEN0
 - hsk_can.c, [131](#)
- BIT_TXEN1
 - hsk_can.c, [131](#)
- BIT_TXPND
 - hsk_can.c, [131](#)
- BIT_TXRQ
 - hsk_can.c, [131](#)
- BIT_TI

- hsk_isr.c, [224](#)
- BIT_TnCLK
 - hsk_pwm.c, [263](#)
- BIT_TnMODEN
 - hsk_pwm.c, [263](#)
- BIT_TnRR
 - hsk_pwm.c, [263](#)
- BIT_TnRS
 - hsk_pwm.c, [263](#)
- BIT_TnSTR
 - hsk_pwm.c, [263](#)
- BIT_VFCTR
 - hsk_adc.c, [96](#)
- BIT_WDTEN
 - hsk_wdt.c, [300](#)
- BIT_WDTIN
 - hsk_wdt.c, [300](#)
- BIT_WDTRS
 - hsk_wdt.c, [300](#)
- BIT_WFR
 - hsk_adc.c, [96](#)
- BIT_XPD
 - hsk_boot.c, [112](#)
- BIT_YE
 - hsk_flash.c, [193](#)
- BYTES_PAGE_DFLASH
 - hsk_flash.c, [193](#)
- BYTES_PAGE_PFLASH
 - hsk_flash.c, [193](#)
- BYTES_WORDLINE_DFLASH
 - hsk_flash.c, [193](#)
- BYTES_WORDLINE_PFLASH
 - hsk_flash.c, [193](#)
- boot
 - hsk_boot.c, [115](#)
 - hsk_flash_struct, [76](#)
- bufState
 - hsk_ssc.c, [284](#)
- buffer
 - hsk_pwc.c, [246](#)
- CAN Node Status Fields, [49](#)
 - CAN_STATUS_ALERT, [49](#)
 - CAN_STATUS_BOFF, [50](#)
 - CAN_STATUS_EWRN, [50](#)
 - CAN_STATUS_LEC, [50](#)
 - CAN_STATUS_RXOK, [50](#)
 - CAN_STATUS_TXOK, [51](#)
- CAN0
 - hsk_can.h, [161](#)
- CAN0_BAUD
 - config.h, [90](#)
- CAN0_IO_P10_P11
 - hsk_can.h, [161](#)
- CAN0_IO_P16_P17
 - hsk_can.h, [161](#)
- CAN0_IO_P34_P35
 - hsk_can.h, [161](#)
- CAN0_IO_P40_P41
 - hsk_can.h, [161](#)
- CAN0_IO
 - config.h, [90](#)
- CAN1
 - hsk_can.h, [161](#)
- CAN1_BAUD
 - config.h, [90](#)
- CAN1_IO_P01_P02
 - hsk_can.h, [161](#)
- CAN1_IO_P14_P13
 - hsk_can.h, [162](#)
- CAN1_IO_P32_P33
 - hsk_can.h, [162](#)
- CAN1_IO
 - config.h, [90](#)
- CAN_AD_READY
 - hsk_can.c, [132](#)
- CAN_AD_READ
 - hsk_can.c, [131](#)
- CAN_AD_WRITE
 - hsk_can.c, [132](#)
- CAN_ENDIAN_INTEL
 - hsk_can.h, [162](#)
- CAN_ENDIAN_MOTOROLA
 - hsk_can.h, [162](#)
- CAN_ERROR
 - hsk_can.h, [162](#)
- CAN_STATUS_ALERT
 - CAN Node Status Fields, [49](#)
- CAN_STATUS_BOFF
 - CAN Node Status Fields, [50](#)
- CAN_STATUS_EWRN
 - CAN Node Status Fields, [50](#)
- CAN_STATUS_LEC
 - CAN Node Status Fields, [50](#)
- CAN_STATUS_RXOK
 - CAN Node Status Fields, [50](#)
- CAN_STATUS_TXOK
 - CAN Node Status Fields, [51](#)
- CANSRC0
 - hsk_isr5_callback, [80](#)
- CANSRC1
 - hsk_isr6_callback, [82](#)
- CANSRC2
 - hsk_isr6_callback, [82](#)
- CANSRC3
 - hsk_isr9_callback, [87](#)
- CCTOVF
 - hsk_isr5_callback, [80](#)
- CHAN_BUF_SIZE
 - hsk_pwc.c, [235](#)
- CLK
 - config.h, [90](#)
- CNT_AM
 - hsk_can.c, [132](#)
- CNT_CCMx
 - hsk_pwc.c, [235](#)
- CNT_CHNR

- hsk_adc.c, [96](#)
- CNT_CTC
 - hsk_adc.c, [96](#)
- CNT_DATA
 - hsk_can.c, [132](#)
- CNT_DLC
 - hsk_can.c, [132](#)
- CNT_EXINTx
 - hsk_pwc.c, [236](#)
- CNT_EXINT
 - hsk_ex.c, [179](#)
- CNT_FILL
 - hsk_adc.c, [96](#)
- CNT_IDEXT
 - hsk_can.c, [132](#)
- CNT_IDSTD
 - hsk_can.c, [133](#)
- CNT_LIST
 - hsk_can.c, [133](#)
- CNT_MMC
 - hsk_can.c, [133](#)
- CNT_MSEL6n
 - hsk_pwm.c, [264](#)
- CNT_MXB
 - hsk_boot.c, [112](#)
- CNT_NDIVH
 - hsk_boot.c, [112](#)
- CNT_NDIVL
 - hsk_boot.c, [112](#)
- CNT_OFVAL
 - hsk_flash.c, [194](#)
- CNT_PDIV
 - hsk_boot.c, [112](#)
- CNT_PRI
 - hsk_can.c, [133](#)
- CNT_PSL
 - hsk_pwm.c, [264](#)
- CNT_REQCHNR
 - hsk_adc.c, [96](#)
- CNT_RESULT
 - hsk_adc.c, [96](#)
- CNT_RXSEL
 - hsk_can.c, [133](#)
- CNT_SEL
 - hsk_ssc.c, [282](#)
- CNT_T0M
 - hsk_timer01.c, [292](#)
- CNT_T1M
 - hsk_timer01.c, [292](#)
- CNT_TnCLK
 - hsk_pwm.c, [264](#)
- CNT_TnMODEN
 - hsk_pwm.c, [264](#)
- callback
 - hsk_timer01.c, [295](#)
- channels
 - hsk_pwc.c, [246](#)
- codepage
 - hsk_icm7228.c, [210](#)
- config.h, [89](#)
 - CAN0_BAUD, [90](#)
 - CAN0_IO, [90](#)
 - CAN1_BAUD, [90](#)
 - CAN1_IO, [90](#)
 - CLK, [90](#)
- DPH
 - hsk_flash.c, [194](#)
- DPL
 - hsk_flash.c, [194](#)
- dflash
 - hsk_flash.c, [199](#)
- dummy
 - hsk_isr.c, [224](#)
- EDGE_DEFAULT_MODE
 - hsk_pwc.c, [236](#)
- EECON
 - hsk_flash.c, [199](#)
- EOFSYN
 - hsk_isr5_callback, [80](#)
- EOC
 - hsk_isr8_callback, [84](#)
- ERRSYN
 - hsk_isr5_callback, [80](#)
- EX_EDGE_BOTH
 - External Interrupt Triggers, [54](#)
- EX_EDGE_DISABLE
 - External Interrupt Triggers, [54](#)
- EX_EDGE_FALLING
 - External Interrupt Triggers, [54](#)
- EX_EDGE_RISING
 - External Interrupt Triggers, [54](#)
- EX_EXINT0
 - External Interrupt Channels, [52](#)
- EX_EXINT0_P05
 - External Interrupt Input Ports, [56](#)
- EX_EXINT0_P14
 - External Interrupt Input Ports, [56](#)
- EX_EXINT1
 - External Interrupt Channels, [52](#)
- EX_EXINT1_P50
 - External Interrupt Input Ports, [56](#)
- EX_EXINT1_P53
 - External Interrupt Input Ports, [56](#)
- EX_EXINT2
 - External Interrupt Channels, [52](#)
- EX_EXINT2_P51
 - External Interrupt Input Ports, [56](#)
- EX_EXINT2_P54
 - External Interrupt Input Ports, [56](#)
- EX_EXINT3
 - External Interrupt Channels, [53](#)
- EX_EXINT3_P11
 - External Interrupt Input Ports, [56](#)
- EX_EXINT3_P30
 - External Interrupt Input Ports, [57](#)

- EX_EXINT3_P40
 - External Interrupt Input Ports, [57](#)
- EX_EXINT3_P55
 - External Interrupt Input Ports, [57](#)
- EX_EXINT4
 - External Interrupt Channels, [53](#)
- EX_EXINT4_P32
 - External Interrupt Input Ports, [57](#)
- EX_EXINT4_P37
 - External Interrupt Input Ports, [57](#)
- EX_EXINT4_P41
 - External Interrupt Input Ports, [57](#)
- EX_EXINT4_P56
 - External Interrupt Input Ports, [57](#)
- EX_EXINT5
 - External Interrupt Channels, [53](#)
- EX_EXINT5_P15
 - External Interrupt Input Ports, [57](#)
- EX_EXINT5_P33
 - External Interrupt Input Ports, [58](#)
- EX_EXINT5_P44
 - External Interrupt Input Ports, [58](#)
- EX_EXINT5_P52
 - External Interrupt Input Ports, [58](#)
- EX_EXINT6
 - External Interrupt Channels, [53](#)
- EX_EXINT6_P16
 - External Interrupt Input Ports, [58](#)
- EX_EXINT6_P34
 - External Interrupt Input Ports, [58](#)
- EX_EXINT6_P42
 - External Interrupt Input Ports, [58](#)
- EX_EXINT6_P45
 - External Interrupt Input Ports, [58](#)
- EX_EXINT6_P57
 - External Interrupt Input Ports, [58](#)
- EXF2
 - [hsk_isr5_callback](#), [80](#)
 - [hsk_isr8_callback](#), [84](#)
- EXINT2
 - [hsk_isr8_callback](#), [84](#)
- EXINT3
 - [hsk_isr9_callback](#), [87](#)
- EXINT4
 - [hsk_isr9_callback](#), [87](#)
- EXINT5
 - [hsk_isr9_callback](#), [87](#)
- EXINT6
 - [hsk_isr9_callback](#), [87](#)
- error
 - [hsk_flash_struct](#), [76](#)
- External Interrupt Channels, [52](#)
 - EX_EXINT0, [52](#)
 - EX_EXINT1, [52](#)
 - EX_EXINT2, [52](#)
 - EX_EXINT3, [53](#)
 - EX_EXINT4, [53](#)
 - EX_EXINT5, [53](#)
 - EX_EXINT6, [53](#)
- External Interrupt Input Ports, [55](#)
 - EX_EXINT0_P05, [56](#)
 - EX_EXINT0_P14, [56](#)
 - EX_EXINT1_P50, [56](#)
 - EX_EXINT1_P53, [56](#)
 - EX_EXINT2_P51, [56](#)
 - EX_EXINT2_P54, [56](#)
 - EX_EXINT3_P11, [56](#)
 - EX_EXINT3_P30, [57](#)
 - EX_EXINT3_P40, [57](#)
 - EX_EXINT3_P55, [57](#)
 - EX_EXINT4_P32, [57](#)
 - EX_EXINT4_P37, [57](#)
 - EX_EXINT4_P41, [57](#)
 - EX_EXINT4_P56, [57](#)
 - EX_EXINT5_P15, [57](#)
 - EX_EXINT5_P33, [58](#)
 - EX_EXINT5_P44, [58](#)
 - EX_EXINT5_P52, [58](#)
 - EX_EXINT6_P16, [58](#)
 - EX_EXINT6_P34, [58](#)
 - EX_EXINT6_P42, [58](#)
 - EX_EXINT6_P45, [58](#)
 - EX_EXINT6_P57, [58](#)
- External Interrupt Triggers, [54](#)
 - EX_EDGE_BOTH, [54](#)
 - EX_EDGE_DISABLE, [54](#)
 - EX_EDGE_FALLING, [54](#)
 - EX_EDGE_RISING, [54](#)
- FCON
 - [hsk_flash.c](#), [199](#)
- FCS1
 - [hsk_flash.c](#), [199](#)
- FCS
 - [hsk_flash.c](#), [199](#)
- FEALH
 - [hsk_flash.c](#), [200](#)
- FEAH
 - [hsk_flash.c](#), [199](#)
- FEAL
 - [hsk_flash.c](#), [199](#)
- FILTER_FACTORY
 - [hsk_filter.h](#), [186](#)
- FILTER_GROUP_FACTORY
 - [hsk_filter.h](#), [187](#)
- FLASH_PWR_FIRST
 - [hsk_flash.h](#), [204](#)
- FLASH_PWR_ON
 - [hsk_flash.h](#), [204](#)
- FLASH_PWR_RESET
 - [hsk_flash.h](#), [205](#)
- FLASH_STRUCT_FACTORY
 - [hsk_flash.h](#), [205](#)
- FREE_BEHIND
 - [hsk_flash.c](#), [194](#)
- FREE_LATEST
 - [hsk_flash.c](#), [194](#)

FREE_NONE
 hsk_flash.c, 194
FTVAL
 hsk_flash.c, 200
flash
 hsk_flash.c, 200
flashDptr
 hsk_flash.c, 200
free
 hsk_flash.c, 200
HSK_CAN_MSG_MAX
 hsk_can.c, 133
hsk_adc.c
 ADC_CHANNELS, 93
 ADC_CLK_12MHz, 93
 ADC_CLK_6MHz, 93
 ADC_CLK_750kHz, 94
 ADC_CLK_8MHz, 94
 ADC_QUEUE, 94
 BIT_ADC_DIS, 94
 BIT_ANON, 94
 BIT_ASEN_PARALLEL, 94
 BIT_ASEN_SEQUENTIAL, 94
 BIT_CHNR, 94
 BIT CTC, 95
 BIT_DW, 95
 BIT_EMPTY, 95
 BIT_ENGT, 95
 BIT_FILL, 95
 BIT_IEN, 95
 BIT_IMODE, 95
 BIT_REQCHNR, 95
 BIT_RESULT, 96
 BIT_VFCTR, 96
 BIT_WFR, 96
 CNT_CHNR, 96
 CNT CTC, 96
 CNT_FILL, 96
 CNT_REQCHNR, 96
 CNT_RESULT, 96
 hsk_adc_close, 97
 hsk_adc_disable, 97
 hsk_adc_enable, 97
 hsk_adc_init, 97
 hsk_adc_isr10, 98
 hsk_adc_isr8, 98
 hsk_adc_isr_warmup10, 98
 hsk_adc_open10, 98
 hsk_adc_open8, 99
 hsk_adc_request, 99
 hsk_adc_service, 99
 hsk_adc_warmup10, 100
 nextChannel, 101
 ptr10, 101
 ptr8, 101
 targets, 101
hsk_adc.h
 ADC_RESOLUTION_10, 103
 ADC_RESOLUTION_8, 103
 hsk_adc_channel, 104
 hsk_adc_close, 104
 hsk_adc_disable, 104
 hsk_adc_enable, 104
 hsk_adc_init, 104
 hsk_adc_open, 103
 hsk_adc_open10, 105
 hsk_adc_open8, 105
 hsk_adc_request, 106
 hsk_adc_service, 106
 hsk_adc_warmup, 103
 hsk_adc_warmup10, 107
hsk_adc/hsk_adc.c, 91
hsk_adc/hsk_adc.h, 101
hsk_adc_channel
 hsk_adc.h, 104
hsk_adc_close
 hsk_adc.c, 97
 hsk_adc.h, 104
hsk_adc_disable
 hsk_adc.c, 97
 hsk_adc.h, 104
hsk_adc_enable
 hsk_adc.c, 97
 hsk_adc.h, 104
hsk_adc_init
 hsk_adc.c, 97
 hsk_adc.h, 104
hsk_adc_isr10
 hsk_adc.c, 98
hsk_adc_isr8
 hsk_adc.c, 98
hsk_adc_isr_warmup10
 hsk_adc.c, 98
hsk_adc_open
 hsk_adc.h, 103
hsk_adc_open10
 hsk_adc.c, 98
 hsk_adc.h, 105
hsk_adc_open8
 hsk_adc.c, 99
 hsk_adc.h, 105
hsk_adc_request
 hsk_adc.c, 99
 hsk_adc.h, 106
hsk_adc_service
 hsk_adc.c, 99
 hsk_adc.h, 106
hsk_adc_warmup
 hsk_adc.h, 103
hsk_adc_warmup10
 hsk_adc.c, 100
 hsk_adc.h, 107
hsk_boot.c
 _sdcc_external_startup, 113
 BIT_EORDRES, 110
 BIT_EXTOSCR, 110

- BIT_MXB19, [110](#)
- BIT_MXB, [110](#)
- BIT_MXM, [110](#)
- BIT_NDIVH, [110](#)
- BIT_NDIVL, [110](#)
- BIT_NMIPLL, [111](#)
- BIT_OSCSS, [111](#)
- BIT_PDIV, [111](#)
- BIT_PLL_LOCK, [111](#)
- BIT_PLLBYP, [111](#)
- BIT_PLLPD, [111](#)
- BIT_PLLRDRES, [112](#)
- BIT_PLLR, [112](#)
- BIT_XPD, [112](#)
- boot, [115](#)
- CNT_MXB, [112](#)
- CNT_NDIVH, [112](#)
- CNT_NDIVL, [112](#)
- CNT_PDIV, [112](#)
- hsk_boot_extClock, [113](#)
- hsk_boot_io, [114](#)
- hsk_boot_isr_nmipll, [115](#)
- hsk_boot_mem, [115](#)
- ndiv, [115](#)
- PDATA_PAGE, [113](#)
- pdiv, [115](#)
- XRAM_BANK, [113](#)
- XRAM_SELECTOR, [113](#)
- hsk_boot.h
 - hsk_boot_extClock, [117](#)
- hsk_boot/hsk_boot.c, [107](#)
- hsk_boot/hsk_boot.h, [116](#)
- hsk_boot_extClock
 - hsk_boot.c, [113](#)
 - hsk_boot.h, [117](#)
- hsk_boot_io
 - hsk_boot.c, [114](#)
- hsk_boot_isr_nmipll
 - hsk_boot.c, [115](#)
- hsk_boot_mem
 - hsk_boot.c, [115](#)
- hsk_can.c
 - AUAD_DEC1, [125](#)
 - AUAD_INC1, [125](#)
 - AUAD_INC8, [126](#)
 - AUAD_OFF, [126](#)
 - BIT_ALIE, [126](#)
 - BIT_AUAD, [126](#)
 - BIT_AM, [126](#)
 - BIT_BRP, [126](#)
 - BIT_BSY, [126](#)
 - BIT_BUSY, [126](#)
 - BIT_CALM, [127](#)
 - BIT_CAN_DIS, [127](#)
 - BIT_CANDIS, [127](#)
 - BIT_CCE, [127](#)
 - BIT_DATA, [127](#)
 - BIT_DIV8, [127](#)
 - BIT_DIR, [127](#)
 - BIT_DLC, [128](#)
 - BIT_ERR, [128](#)
 - BIT_FCCFG, [128](#)
 - BIT_IDEXT, [128](#)
 - BIT_IDSTD, [128](#)
 - BIT_IDE, [128](#)
 - BIT_INIT, [128](#)
 - BIT_LECIE, [129](#)
 - BIT_LIST, [129](#)
 - BIT_MIDE, [129](#)
 - BIT_MMC, [129](#)
 - BIT_MSGVAL, [129](#)
 - BIT_NEWDAT, [129](#)
 - BIT_PRI, [129](#)
 - BIT_RBUSY, [129](#)
 - BIT_RWEN, [130](#)
 - BIT_RXEN, [130](#)
 - BIT_RXPND, [130](#)
 - BIT_RXSEL, [130](#)
 - BIT_RXUPD, [130](#)
 - BIT_SJW, [130](#)
 - BIT_TRIE, [130](#)
 - BIT_TSEG1, [131](#)
 - BIT_TSEG2, [131](#)
 - BIT_TXEN0, [131](#)
 - BIT_TXEN1, [131](#)
 - BIT_TXPND, [131](#)
 - BIT_TXRQ, [131](#)
 - CAN_AD_READY, [132](#)
 - CAN_AD_READ, [131](#)
 - CAN_AD_WRITE, [132](#)
 - CNT_AM, [132](#)
 - CNT_DATA, [132](#)
 - CNT_DLC, [132](#)
 - CNT_IDEXT, [132](#)
 - CNT_IDSTD, [133](#)
 - CNT_LIST, [133](#)
 - CNT_MMC, [133](#)
 - CNT_PRI, [133](#)
 - CNT_RXSEL, [133](#)
 - HSK_CAN_MSG_MAX, [133](#)
 - hsk_can_data_getIntelSignal, [140](#)
 - hsk_can_data_getMotorolaSignal, [141](#)
 - hsk_can_data_getSignal, [141](#)
 - hsk_can_data_setIntelSignal, [142](#)
 - hsk_can_data_setMotorolaSignal, [142](#)
 - hsk_can_data_setSignal, [143](#)
 - hsk_can_disable, [144](#)
 - hsk_can_enable, [144](#)
 - hsk_can_fifo_connect, [144](#)
 - hsk_can_fifo_create, [145](#)
 - hsk_can_fifo_delete, [146](#)
 - hsk_can_fifo_disconnect, [146](#)
 - hsk_can_fifo_getData, [147](#)
 - hsk_can_fifo_getId, [147](#)
 - hsk_can_fifo_move, [148](#)
 - hsk_can_fifo_next, [148](#)

- hsk_can_fifo_setRxMask, 148
- hsk_can_fifo_setupRx, 149
- hsk_can_fifo_updated, 149
- hsk_can_init, 150
- hsk_can_msg_connect, 151
- hsk_can_msg_create, 152
- hsk_can_msg_delete, 152
- hsk_can_msg_disconnect, 153
- hsk_can_msg_getData, 153
- hsk_can_msg_move, 154
- hsk_can_msg_receive, 154
- hsk_can_msg_send, 154
- hsk_can_msg_sent, 155
- hsk_can_msg_setData, 155
- hsk_can_msg_updated, 155
- hsk_can_status, 156
- initialised, 156
- LIST_NODEx, 133
- LIST_PENDING, 133
- LIST_UNALLOC, 134
- MMC_DEFAULT, 134
- MMC_GATEWAYSRC, 134
- MMC_RXBASEFIFO, 134
- MMC_TXBASEFIFO, 134
- MMC_TXSLAVEFIFO, 134
- MOAMRn, 134
- MOARn, 134
- MOCTRn, 135
- MODATAHn, 135
- MODATALn, 135
- MOFCRn, 135
- MOFGPRn, 135
- MOFGPRn_BOT, 135
- MOFGPRn_CUR, 135
- MOFGPRn_SEL, 135
- MOFGPRn_TOP, 136
- MOSTATn, 136
- MOSTATn_PNEXT, 136
- NBTRx, 136
- NCRx, 136
- NECNTx, 136
- NFCRx, 136
- NIPRx, 137
- NPCRx, 137
- NSRx, 137
- OFF_LISTm, 137
- OFF_MOn, 137
- OFF_MSIDk, 137
- OFF_MSPNDk, 137
- OFF_NODEx, 137
- PAN_CMD_ALLOCBEFORE, 138
- PAN_CMD_ALLOCBEHIND, 138
- PAN_CMD_ALLOC, 138
- PAN_CMD_INIT, 138
- PAN_CMD_MOVEBEFORE, 138
- PAN_CMD_MOVEBEHIND, 138
- PAN_CMD_MOVE, 138
- PAN_CMD_NOP, 138
- PANAR1, 139
- PANAR2, 139
- PANCMD, 139
- PANCTR_READY, 139
- PANCTR, 139
- PANSTATUS, 139
- PRI_ID, 140
- PRI_LIST, 140
- RESET_DATA, 140
- RESET, 140
- SET_DATA, 140
- SET, 140
- hsk_can.h
 - CAN0, 161
 - CAN0_IO_P10_P11, 161
 - CAN0_IO_P16_P17, 161
 - CAN0_IO_P34_P35, 161
 - CAN0_IO_P40_P41, 161
 - CAN1, 161
 - CAN1_IO_P01_P02, 161
 - CAN1_IO_P14_P13, 162
 - CAN1_IO_P32_P33, 162
 - CAN_ENDIAN_INTEL, 162
 - CAN_ENDIAN_MOTOROLA, 162
 - CAN_ERROR, 162
 - hsk_can_data_getSignal, 163
 - hsk_can_data_setSignal, 164
 - hsk_can_disable, 165
 - hsk_can_enable, 165
 - hsk_can_fifo, 163
 - hsk_can_fifo_connect, 165
 - hsk_can_fifo_create, 166
 - hsk_can_fifo_delete, 167
 - hsk_can_fifo_disconnect, 167
 - hsk_can_fifo_getData, 168
 - hsk_can_fifo_getId, 168
 - hsk_can_fifo_next, 169
 - hsk_can_fifo_setRxMask, 169
 - hsk_can_fifo_setupRx, 169
 - hsk_can_fifo_updated, 170
 - hsk_can_init, 170
 - hsk_can_msg, 163
 - hsk_can_msg_connect, 171
 - hsk_can_msg_create, 172
 - hsk_can_msg_delete, 173
 - hsk_can_msg_disconnect, 173
 - hsk_can_msg_getData, 174
 - hsk_can_msg_receive, 174
 - hsk_can_msg_send, 174
 - hsk_can_msg_sent, 175
 - hsk_can_msg_setData, 175
 - hsk_can_msg_updated, 175
 - hsk_can_node, 163
 - hsk_can_status, 176
- hsk_can/hsk_can.c, 118
- hsk_can/hsk_can.h, 157
- hsk_can_data_getIntelSignal
 - hsk_can.c, 140

- hsk_can_data_getMotorolaSignal
 - hsk_can.c, [141](#)
- hsk_can_data_getSignal
 - hsk_can.c, [141](#)
 - hsk_can.h, [163](#)
- hsk_can_data_setIntelSignal
 - hsk_can.c, [142](#)
- hsk_can_data_setMotorolaSignal
 - hsk_can.c, [142](#)
- hsk_can_data_setSignal
 - hsk_can.c, [143](#)
 - hsk_can.h, [164](#)
- hsk_can_disable
 - hsk_can.c, [144](#)
 - hsk_can.h, [165](#)
- hsk_can_enable
 - hsk_can.c, [144](#)
 - hsk_can.h, [165](#)
- hsk_can_fifo
 - hsk_can.h, [163](#)
- hsk_can_fifo_connect
 - hsk_can.c, [144](#)
 - hsk_can.h, [165](#)
- hsk_can_fifo_create
 - hsk_can.c, [145](#)
 - hsk_can.h, [166](#)
- hsk_can_fifo_delete
 - hsk_can.c, [146](#)
 - hsk_can.h, [167](#)
- hsk_can_fifo_disconnect
 - hsk_can.c, [146](#)
 - hsk_can.h, [167](#)
- hsk_can_fifo_getData
 - hsk_can.c, [147](#)
 - hsk_can.h, [168](#)
- hsk_can_fifo_getId
 - hsk_can.c, [147](#)
 - hsk_can.h, [168](#)
- hsk_can_fifo_move
 - hsk_can.c, [148](#)
- hsk_can_fifo_next
 - hsk_can.c, [148](#)
 - hsk_can.h, [169](#)
- hsk_can_fifo_setRxMask
 - hsk_can.c, [148](#)
 - hsk_can.h, [169](#)
- hsk_can_fifo_setupRx
 - hsk_can.c, [149](#)
 - hsk_can.h, [169](#)
- hsk_can_fifo_updated
 - hsk_can.c, [149](#)
 - hsk_can.h, [170](#)
- hsk_can_init
 - hsk_can.c, [150](#)
 - hsk_can.h, [170](#)
- hsk_can_msg
 - hsk_can.h, [163](#)
- hsk_can_msg_connect
 - hsk_can.c, [151](#)
 - hsk_can.h, [171](#)
- hsk_can_msg_create
 - hsk_can.c, [152](#)
 - hsk_can.h, [172](#)
- hsk_can_msg_delete
 - hsk_can.c, [152](#)
 - hsk_can.h, [173](#)
- hsk_can_msg_disconnect
 - hsk_can.c, [153](#)
 - hsk_can.h, [173](#)
- hsk_can_msg_getData
 - hsk_can.c, [153](#)
 - hsk_can.h, [174](#)
- hsk_can_msg_move
 - hsk_can.c, [154](#)
- hsk_can_msg_receive
 - hsk_can.c, [154](#)
 - hsk_can.h, [174](#)
- hsk_can_msg_send
 - hsk_can.c, [154](#)
 - hsk_can.h, [174](#)
- hsk_can_msg_sent
 - hsk_can.c, [155](#)
 - hsk_can.h, [175](#)
- hsk_can_msg_setData
 - hsk_can.c, [155](#)
 - hsk_can.h, [175](#)
- hsk_can_msg_updated
 - hsk_can.c, [155](#)
 - hsk_can.h, [175](#)
- hsk_can_node
 - hsk_can.h, [163](#)
- hsk_can_status
 - hsk_can.c, [156](#)
 - hsk_can.h, [176](#)
- hsk_ex.c
 - BIT_EXINT0, [178](#)
 - BIT_EXINT1, [178](#)
 - BIT_EXINT2, [178](#)
 - BIT_EXINT3, [178](#)
 - BIT_EXINT4, [178](#)
 - BIT_EXINT5, [179](#)
 - BIT_EXINT6, [179](#)
 - BIT_IMODE, [179](#)
 - CNT_EXINT, [179](#)
 - hsk_ex_channel_disable, [179](#)
 - hsk_ex_channel_enable, [179](#)
 - hsk_ex_port_close, [180](#)
 - hsk_ex_port_open, [180](#)
 - modpiselBit, [180](#)
 - modpiselSel, [181](#)
 - portAltSel, [181](#)
 - portBit, [181](#)
 - ports, [181](#)
- hsk_ex.h
 - hsk_ex_channel, [184](#)
 - hsk_ex_channel_disable, [184](#)

- hsk_ex_channel_enable, 185
- hsk_ex_port, 184
- hsk_ex_port_close, 185
- hsk_ex_port_open, 185
- hsk_ex/hsk_ex.c, 176
- hsk_ex/hsk_ex.h, 182
- hsk_ex_channel
 - hsk_ex.h, 184
- hsk_ex_channel_disable
 - hsk_ex.c, 179
 - hsk_ex.h, 184
- hsk_ex_channel_enable
 - hsk_ex.c, 179
 - hsk_ex.h, 185
- hsk_ex_port
 - hsk_ex.h, 184
- hsk_ex_port_close
 - hsk_ex.c, 180
 - hsk_ex.h, 185
- hsk_ex_port_open
 - hsk_ex.c, 180
 - hsk_ex.h, 185
- hsk_filter.h
 - FILTER_FACTORY, 186
 - FILTER_GROUP_FACTORY, 187
- hsk_filter/hsk_filter.h, 186
- hsk_flash.c
 - ADDR_DFLASH, 191
 - ADDR_PFLASH, 191
 - ADDR_ROM, 191
 - ADDR_XRAM, 191
 - BIT_EEABORT, 192
 - BIT_EEBSY, 192
 - BIT_ERASE, 192
 - BIT_FTEN, 192
 - BIT_MAS1, 192
 - BIT_MODE, 192
 - BIT_NMIFLASH, 193
 - BIT_NVSTR, 193
 - BIT_OFVAL, 193
 - BIT_PROG, 193
 - BIT_YE, 193
 - BYTES_PAGE_DFLASH, 193
 - BYTES_PAGE_PFLASH, 193
 - BYTES_WORDLINE_DFLASH, 193
 - BYTES_WORDLINE_PFLASH, 193
 - CNT_OFVAL, 194
 - DPH, 194
 - DPL, 194
 - dflash, 199
 - EECON, 199
 - FCON, 199
 - FCS1, 199
 - FCS, 199
 - FEALH, 200
 - FEAH, 199
 - FEAL, 199
 - FREE_BEHIND, 194
 - FREE_LATEST, 194
 - FREE_NONE, 194
 - FTVAL, 200
 - flash, 200
 - flashDptr, 200
 - free, 200
 - hsk_flash_init, 197
 - hsk_flash_isr_nmiflash, 197
 - hsk_flash_write, 198
 - ident, 200
 - LEN_DFLASH, 195
 - LEN_PFLASH, 195
 - LEN_ROM, 195
 - LEN_XRAM, 195
 - latest, 201
 - MOVCI, 195
 - oldest, 201
 - PAGE_FLASH, 195
 - PAGE_RAM, 195
 - ptr, 201
 - STATE_DELETE, 195
 - STATE_DETECT, 196
 - STATE_IDLE, 196
 - STATE_REQUEST, 196
 - STATE_RESET, 196
 - STATE_WRITE, 196
 - size, 201
 - state, 201
 - VAR_ASM, 196
 - VAR_AT, 196
 - wrap, 201
 - xdataDptr, 201
- hsk_flash.h
 - FLASH_PWR_FIRST, 204
 - FLASH_PWR_ON, 204
 - FLASH_PWR_RESET, 205
 - FLASH_STRUCT_FACTORY, 205
 - hsk_flash_init, 206
 - hsk_flash_write, 207
 - XC878_16FF, 206
- hsk_flash/hsk_flash.c, 187
- hsk_flash/hsk_flash.h, 202
- hsk_flash_chksum
 - hsk_flash_struct, 76
- hsk_flash_init
 - hsk_flash.c, 197
 - hsk_flash.h, 206
- hsk_flash_isr_nmiflash
 - hsk_flash.c, 197
- hsk_flash_prefix
 - hsk_flash_struct, 76
- hsk_flash_struct, 75
 - boot, 76
 - error, 76
 - hsk_flash_chksum, 76
 - hsk_flash_prefix, 76
 - reset, 76
- hsk_flash_write

- hsk_flash.c, [198](#)
- hsk_flash.h, [207](#)
- hsk_icm7228.c
 - codepage, [210](#)
 - hsk_icm7228_illuminate, [208](#)
 - hsk_icm7228_writeDec, [209](#)
 - hsk_icm7228_writeHex, [209](#)
 - hsk_icm7228_writeString, [210](#)
 - ILLUMINATE_OFFSET, [208](#)
- hsk_icm7228.h
 - hsk_icm7228_illuminate, [213](#)
 - hsk_icm7228_writeDec, [213](#)
 - hsk_icm7228_writeHex, [214](#)
 - hsk_icm7228_writeString, [214](#)
 - ICM7228_FACTORY, [212](#)
- hsk_icm7228/hsk_icm7228.c, [207](#)
- hsk_icm7228/hsk_icm7228.h, [211](#)
- hsk_icm7228_illuminate
 - hsk_icm7228.c, [208](#)
 - hsk_icm7228.h, [213](#)
- hsk_icm7228_writeDec
 - hsk_icm7228.c, [209](#)
 - hsk_icm7228.h, [213](#)
- hsk_icm7228_writeHex
 - hsk_icm7228.c, [209](#)
 - hsk_icm7228.h, [214](#)
- hsk_icm7228_writeString
 - hsk_icm7228.c, [210](#)
 - hsk_icm7228.h, [214](#)
- hsk_io/hsk_io.h, [215](#)
- hsk_isr.c
 - BIT_ADCSR0, [220](#)
 - BIT_ADCSR1, [220](#)
 - BIT_CANSRC0, [220](#)
 - BIT_CANSRC1, [220](#)
 - BIT_CANSRC2, [220](#)
 - BIT_CANSRC3, [220](#)
 - BIT_CCTOVF, [220](#)
 - BIT_EOFSYN, [221](#)
 - BIT_EOC, [221](#)
 - BIT_ERRSYN, [221](#)
 - BIT_EXF2, [221](#)
 - BIT_EXINT2, [221](#)
 - BIT_EXINT3, [221](#)
 - BIT_EXINT4, [222](#)
 - BIT_EXINT5, [222](#)
 - BIT_EXINT6, [222](#)
 - BIT_IERR, [222](#)
 - BIT_IRDY, [222](#)
 - BIT_NDOV, [222](#)
 - BIT_NMIECC, [223](#)
 - BIT_NMIFLASH, [223](#)
 - BIT_NMIPLL, [223](#)
 - BIT_NMIVDDP, [223](#)
 - BIT_NMIWDT, [223](#)
 - BIT_RMAP, [223](#)
 - BIT_RI, [223](#)
 - BIT_TF2, [224](#)
- BIT_TI, [224](#)
- dummy, [224](#)
- hsk_isr14, [227](#)
- hsk_isr5, [227](#)
- hsk_isr6, [227](#)
- hsk_isr8, [227](#)
- hsk_isr9, [227](#)
- hsk_isr_root1, [224](#)
- ISR_hsk_isr14, [224](#)
- ISR_hsk_isr5, [225](#)
- ISR_hsk_isr6, [225](#)
- ISR_hsk_isr8, [225](#)
- ISR_hsk_isr9, [226](#)
- nmidummy, [226](#)
- hsk_isr.h
 - hsk_isr14, [230](#)
 - hsk_isr5, [230](#)
 - hsk_isr6, [230](#)
 - hsk_isr8, [230](#)
 - hsk_isr9, [230](#)
- hsk_isr/hsk_isr.c, [217](#)
- hsk_isr/hsk_isr.h, [228](#)
- hsk_isr14
 - hsk_isr.c, [227](#)
 - hsk_isr.h, [230](#)
- hsk_isr14_callback, [77](#)
- NMIECC, [78](#)
- NMIFLASH, [78](#)
- NMIPLL, [78](#)
- NMIVDDP, [78](#)
- NMIWDT, [78](#)
- hsk_isr5
 - hsk_isr.c, [227](#)
 - hsk_isr.h, [230](#)
- hsk_isr5_callback, [79](#)
- CANSRC0, [80](#)
- CCTOVF, [80](#)
- EOFSYN, [80](#)
- ERRSYN, [80](#)
- EXF2, [80](#)
- NDOV, [81](#)
- TF2, [81](#)
- hsk_isr6
 - hsk_isr.c, [227](#)
 - hsk_isr.h, [230](#)
- hsk_isr6_callback, [81](#)
- ADCSR0, [82](#)
- ADCSR1, [82](#)
- CANSRC1, [82](#)
- CANSRC2, [82](#)
- hsk_isr8
 - hsk_isr.c, [227](#)
 - hsk_isr.h, [230](#)
- hsk_isr8_callback, [83](#)
- EOC, [84](#)
- EXF2, [84](#)
- EXINT2, [84](#)
- IERR, [85](#)

- IRDY, [85](#)
- NDOV, [85](#)
- RI, [85](#)
- TF2, [85](#)
- TI, [85](#)
- hsk_isr9
 - hsk_isr.c, [227](#)
 - hsk_isr.h, [230](#)
- hsk_isr9_callback, [86](#)
 - CANSRC3, [87](#)
 - EXINT3, [87](#)
 - EXINT4, [87](#)
 - EXINT5, [87](#)
 - EXINT6, [87](#)
- hsk_isr_root1
 - hsk_isr.c, [224](#)
- hsk_pwc.c
 - averageOver, [246](#)
 - BIT_CCM0, [234](#)
 - BIT_CCTBx, [234](#)
 - BIT_CCTOVEN, [234](#)
 - BIT_CCTOVF, [234](#)
 - BIT_CCTPRE, [234](#)
 - BIT_CCTST, [235](#)
 - BIT_IMODE, [235](#)
 - BIT_T2CCFG, [235](#)
 - BIT_T2CCU_DIS, [235](#)
 - BIT_TIMSYN, [235](#)
 - buffer, [246](#)
 - CHAN_BUF_SIZE, [235](#)
 - CNT_CCMx, [235](#)
 - CNT_EXINTx, [236](#)
 - channels, [246](#)
 - EDGE_DEFAULT_MODE, [236](#)
 - hsk_pwc_ccn, [237](#)
 - hsk_pwc_channel_captureMode, [237](#)
 - hsk_pwc_channel_close, [238](#)
 - hsk_pwc_channel_edgeMode, [238](#)
 - hsk_pwc_channel_getValue, [238](#)
 - hsk_pwc_channel_open, [239](#)
 - hsk_pwc_channel_trigger, [239](#)
 - hsk_pwc_disable, [240](#)
 - hsk_pwc_enable, [240](#)
 - hsk_pwc_init, [240](#)
 - hsk_pwc_isr_cc0_p30, [241](#)
 - hsk_pwc_isr_cc0_p40, [241](#)
 - hsk_pwc_isr_cc0_p55, [241](#)
 - hsk_pwc_isr_cc1_p32, [242](#)
 - hsk_pwc_isr_cc1_p41, [242](#)
 - hsk_pwc_isr_cc1_p56, [242](#)
 - hsk_pwc_isr_cc2_p33, [243](#)
 - hsk_pwc_isr_cc2_p44, [243](#)
 - hsk_pwc_isr_cc2_p52, [243](#)
 - hsk_pwc_isr_cc3_p34, [244](#)
 - hsk_pwc_isr_cc3_p45, [244](#)
 - hsk_pwc_isr_cc3_p57, [244](#)
 - hsk_pwc_isr_ccn, [245](#)
 - hsk_pwc_isr_cctOverflow, [245](#)
 - hsk_pwc_port_open, [245](#)
 - hsk_pwc_ports, [247](#)
 - inBit, [247](#)
 - inCount, [247](#)
 - inSel, [247](#)
 - invalid, [247](#)
 - lastCapture, [247](#)
 - overflow, [248](#)
 - overflows, [248](#)
 - PWC_CC0_EXINT_BIT, [236](#)
 - PWC_CC0_EXINT_REG, [236](#)
 - PWC_CC1_EXINT_BIT, [236](#)
 - PWC_CC1_EXINT_REG, [236](#)
 - PWC_CC2_EXINT_BIT, [236](#)
 - PWC_CC2_EXINT_REG, [236](#)
 - PWC_CC3_EXINT_BIT, [237](#)
 - PWC_CC3_EXINT_REG, [237](#)
 - PWC_CHANNELS, [237](#)
 - portBit, [248](#)
 - portSel, [248](#)
 - pos, [248](#)
 - prescaler, [248](#)
 - state, [248](#)
 - sum, [249](#)
- hsk_pwc.h
 - hsk_pwc_channel, [255](#)
 - hsk_pwc_channel_captureMode, [255](#)
 - hsk_pwc_channel_close, [255](#)
 - hsk_pwc_channel_edgeMode, [256](#)
 - hsk_pwc_channel_getValue, [256](#)
 - hsk_pwc_channel_open, [256](#)
 - hsk_pwc_channel_trigger, [257](#)
 - hsk_pwc_disable, [258](#)
 - hsk_pwc_enable, [258](#)
 - hsk_pwc_init, [258](#)
 - hsk_pwc_port, [255](#)
 - hsk_pwc_port_open, [259](#)
 - PWC_CC0, [252](#)
 - PWC_CC0_P30, [252](#)
 - PWC_CC0_P40, [252](#)
 - PWC_CC0_P55, [252](#)
 - PWC_CC1, [253](#)
 - PWC_CC1_P32, [253](#)
 - PWC_CC1_P41, [253](#)
 - PWC_CC1_P56, [253](#)
 - PWC_CC2, [253](#)
 - PWC_CC2_P33, [253](#)
 - PWC_CC2_P44, [253](#)
 - PWC_CC2_P52, [253](#)
 - PWC_CC3, [254](#)
 - PWC_CC3_P34, [254](#)
 - PWC_CC3_P45, [254](#)
 - PWC_CC3_P57, [254](#)
 - PWC_EDGE_BOTH, [254](#)
 - PWC_EDGE_FALLING, [254](#)
 - PWC_EDGE_RISING, [254](#)
 - PWC_MODE_EXT, [254](#)
 - PWC_MODE_SOFT, [255](#)

- hsk_pwc/hsk_pwc.c, [231](#)
- hsk_pwc/hsk_pwc.h, [249](#)
- hsk_pwc_ccn
 - hsk_pwc.c, [237](#)
- hsk_pwc_channel
 - hsk_pwc.h, [255](#)
- hsk_pwc_channel_captureMode
 - hsk_pwc.c, [237](#)
 - hsk_pwc.h, [255](#)
- hsk_pwc_channel_close
 - hsk_pwc.c, [238](#)
 - hsk_pwc.h, [255](#)
- hsk_pwc_channel_edgeMode
 - hsk_pwc.c, [238](#)
 - hsk_pwc.h, [256](#)
- hsk_pwc_channel_getValue
 - hsk_pwc.c, [238](#)
 - hsk_pwc.h, [256](#)
- hsk_pwc_channel_open
 - hsk_pwc.c, [239](#)
 - hsk_pwc.h, [256](#)
- hsk_pwc_channel_trigger
 - hsk_pwc.c, [239](#)
 - hsk_pwc.h, [257](#)
- hsk_pwc_disable
 - hsk_pwc.c, [240](#)
 - hsk_pwc.h, [258](#)
- hsk_pwc_enable
 - hsk_pwc.c, [240](#)
 - hsk_pwc.h, [258](#)
- hsk_pwc_init
 - hsk_pwc.c, [240](#)
 - hsk_pwc.h, [258](#)
- hsk_pwc_isr_cc0_p30
 - hsk_pwc.c, [241](#)
- hsk_pwc_isr_cc0_p40
 - hsk_pwc.c, [241](#)
- hsk_pwc_isr_cc0_p55
 - hsk_pwc.c, [241](#)
- hsk_pwc_isr_cc1_p32
 - hsk_pwc.c, [242](#)
- hsk_pwc_isr_cc1_p41
 - hsk_pwc.c, [242](#)
- hsk_pwc_isr_cc1_p56
 - hsk_pwc.c, [242](#)
- hsk_pwc_isr_cc2_p33
 - hsk_pwc.c, [243](#)
- hsk_pwc_isr_cc2_p44
 - hsk_pwc.c, [243](#)
- hsk_pwc_isr_cc2_p52
 - hsk_pwc.c, [243](#)
- hsk_pwc_isr_cc3_p34
 - hsk_pwc.c, [244](#)
- hsk_pwc_isr_cc3_p45
 - hsk_pwc.c, [244](#)
- hsk_pwc_isr_cc3_p57
 - hsk_pwc.c, [244](#)
- hsk_pwc_isr_ccn
 - hsk_pwc.c, [245](#)
- hsk_pwc_isr_cctOverflow
 - hsk_pwc.c, [245](#)
- hsk_pwc_port
 - hsk_pwc.h, [255](#)
- hsk_pwc_port_open
 - hsk_pwc.c, [245](#)
 - hsk_pwc.h, [259](#)
- hsk_pwc_ports
 - hsk_pwc.c, [247](#)
- hsk_pwm.c
 - BIT_CCUCFG, [262](#)
 - BIT_CCUCFG, [262](#)
 - BIT_ECT13O, [263](#)
 - BIT_PSL63, [263](#)
 - BIT_PSL, [263](#)
 - BIT_TnCLK, [263](#)
 - BIT_TnMODEN, [263](#)
 - BIT_TnRR, [263](#)
 - BIT_TnRS, [263](#)
 - BIT_TnSTR, [263](#)
 - CNT_MSEL6n, [264](#)
 - CNT_PSL, [264](#)
 - CNT_TnCLK, [264](#)
 - CNT_TnMODEN, [264](#)
 - hsk_pwm_channel_set, [264](#)
 - hsk_pwm_disable, [265](#)
 - hsk_pwm_enable, [265](#)
 - hsk_pwm_init, [265](#)
 - hsk_pwm_outChannel_dir, [266](#)
 - hsk_pwm_port_close, [266](#)
 - hsk_pwm_port_open, [267](#)
 - MOD_MSEL6n, [264](#)
 - ports, [267](#)
 - pos, [267](#)
 - sel, [268](#)
- hsk_pwm.h
 - hsk_pwm_channel, [275](#)
 - hsk_pwm_channel_set, [275](#)
 - hsk_pwm_disable, [276](#)
 - hsk_pwm_enable, [276](#)
 - hsk_pwm_init, [276](#)
 - hsk_pwm_outChannel, [275](#)
 - hsk_pwm_outChannel_dir, [277](#)
 - hsk_pwm_port, [275](#)
 - hsk_pwm_port_close, [277](#)
 - hsk_pwm_port_open, [278](#)
 - PWM_60, [271](#)
 - PWM_61, [271](#)
 - PWM_62, [271](#)
 - PWM_63, [271](#)
 - PWM_CC60, [271](#)
 - PWM_CC61, [271](#)
 - PWM_CC62, [272](#)
 - PWM_COUT60, [272](#)
 - PWM_COUT61, [272](#)
 - PWM_COUT62, [272](#)
 - PWM_COUT63, [272](#)

- PWM_OUT_60_P30, [272](#)
- PWM_OUT_60_P31, [272](#)
- PWM_OUT_60_P40, [272](#)
- PWM_OUT_60_P41, [273](#)
- PWM_OUT_61_P00, [273](#)
- PWM_OUT_61_P01, [273](#)
- PWM_OUT_61_P31, [273](#)
- PWM_OUT_61_P32, [273](#)
- PWM_OUT_61_P33, [273](#)
- PWM_OUT_61_P44, [273](#)
- PWM_OUT_61_P45, [273](#)
- PWM_OUT_62_P04, [274](#)
- PWM_OUT_62_P05, [274](#)
- PWM_OUT_62_P34, [274](#)
- PWM_OUT_62_P35, [274](#)
- PWM_OUT_62_P46, [274](#)
- PWM_OUT_62_P47, [274](#)
- PWM_OUT_63_P03, [274](#)
- PWM_OUT_63_P37, [274](#)
- PWM_OUT_63_P43, [275](#)
- hsk_pwm/hsk_pwm.c, [260](#)
- hsk_pwm/hsk_pwm.h, [268](#)
- hsk_pwm_channel
 - hsk_pwm.h, [275](#)
- hsk_pwm_channel_set
 - hsk_pwm.c, [264](#)
 - hsk_pwm.h, [275](#)
- hsk_pwm_disable
 - hsk_pwm.c, [265](#)
 - hsk_pwm.h, [276](#)
- hsk_pwm_enable
 - hsk_pwm.c, [265](#)
 - hsk_pwm.h, [276](#)
- hsk_pwm_init
 - hsk_pwm.c, [265](#)
 - hsk_pwm.h, [276](#)
- hsk_pwm_outChannel
 - hsk_pwm.h, [275](#)
- hsk_pwm_outChannel_dir
 - hsk_pwm.c, [266](#)
 - hsk_pwm.h, [277](#)
- hsk_pwm_port
 - hsk_pwm.h, [275](#)
- hsk_pwm_port_close
 - hsk_pwm.c, [266](#)
 - hsk_pwm.h, [277](#)
- hsk_pwm_port_open
 - hsk_pwm.c, [267](#)
 - hsk_pwm.h, [278](#)
- hsk_ssc.c
 - BIT_CIS, [280](#)
 - BIT_EIREN, [280](#)
 - BIT_EIR, [280](#)
 - BIT_EN, [280](#)
 - BIT_LB, [281](#)
 - BIT_MIS, [281](#)
 - BIT_MS, [281](#)
 - BIT_RIREN, [281](#)
 - BIT_RIR, [281](#)
 - BIT_RMAP, [281](#)
 - BIT_SIS, [281](#)
 - BIT_SSC_DIS, [282](#)
 - BIT_TIREN, [282](#)
 - BIT_TIR, [282](#)
 - bufState, [284](#)
 - CNT_SEL, [282](#)
 - hsk_ssc_disable, [282](#)
 - hsk_ssc_enable, [282](#)
 - hsk_ssc_init, [282](#)
 - hsk_ssc_ports, [283](#)
 - hsk_ssc_talk, [283](#)
 - ISR_hsk_ssc, [283](#)
 - rcount, [284](#)
 - rprr, [284](#)
 - wcount, [284](#)
 - wptr, [284](#)
- hsk_ssc.h
 - hsk_ssc_busy, [287](#)
 - hsk_ssc_disable, [288](#)
 - hsk_ssc_enable, [288](#)
 - hsk_ssc_init, [289](#)
 - hsk_ssc_ports, [289](#)
 - hsk_ssc_talk, [289](#)
 - SSC_BAUD, [287](#)
 - SSC_CONF, [288](#)
 - SSC_MASTER, [288](#)
 - SSC_SLAVE, [288](#)
- hsk_ssc/hsk_ssc.c, [278](#)
- hsk_ssc/hsk_ssc.h, [285](#)
- hsk_ssc_busy
 - hsk_ssc.h, [287](#)
- hsk_ssc_disable
 - hsk_ssc.c, [282](#)
 - hsk_ssc.h, [288](#)
- hsk_ssc_enable
 - hsk_ssc.c, [282](#)
 - hsk_ssc.h, [288](#)
- hsk_ssc_init
 - hsk_ssc.c, [282](#)
 - hsk_ssc.h, [289](#)
- hsk_ssc_ports
 - hsk_ssc.c, [283](#)
 - hsk_ssc.h, [289](#)
- hsk_ssc_talk
 - hsk_ssc.c, [283](#)
 - hsk_ssc.h, [289](#)
- hsk_timer01.c
 - BIT_ET0, [292](#)
 - BIT_ET1, [292](#)
 - BIT_RMAP, [292](#)
 - BIT_T0M, [292](#)
 - BIT_T1M, [292](#)
 - CNT_T0M, [292](#)
 - CNT_T1M, [292](#)
 - callback, [295](#)
 - hsk_timer01_setup, [292](#)

- hsk_timer0_disable, 293
- hsk_timer0_enable, 293
- hsk_timer0_setup, 293
- hsk_timer1_disable, 294
- hsk_timer1_enable, 294
- hsk_timer1_setup, 294
- ISR_hsk_timer0, 295
- ISR_hsk_timer1, 295
- overflow, 295
- timers, 295
- hsk_timer01.h
 - hsk_timer0_disable, 297
 - hsk_timer0_enable, 297
 - hsk_timer0_setup, 297
 - hsk_timer1_disable, 298
 - hsk_timer1_enable, 298
 - hsk_timer1_setup, 298
- hsk_timer01_setup
 - hsk_timer01.c, 292
- hsk_timer0_disable
 - hsk_timer01.c, 293
 - hsk_timer01.h, 297
- hsk_timer0_enable
 - hsk_timer01.c, 293
 - hsk_timer01.h, 297
- hsk_timer0_setup
 - hsk_timer01.c, 293
 - hsk_timer01.h, 297
- hsk_timer1_disable
 - hsk_timer01.c, 294
 - hsk_timer01.h, 298
- hsk_timer1_enable
 - hsk_timer01.c, 294
 - hsk_timer01.h, 298
- hsk_timer1_setup
 - hsk_timer01.c, 294
 - hsk_timer01.h, 298
- hsk_timers/hsk_timer01.c, 290
- hsk_timers/hsk_timer01.h, 296
- hsk_wdt.c
 - BIT_WDTEN, 300
 - BIT_WDTIN, 300
 - BIT_WDTRS, 300
 - hsk_wdt_disable, 301
 - hsk_wdt_enable, 301
 - hsk_wdt_init, 301
 - hsk_wdt_service, 301
- hsk_wdt.h
 - hsk_wdt_disable, 303
 - hsk_wdt_enable, 303
 - hsk_wdt_init, 303
 - hsk_wdt_service, 304
- hsk_wdt/hsk_wdt.c, 299
- hsk_wdt/hsk_wdt.h, 302
- hsk_wdt_disable
 - hsk_wdt.c, 301
 - hsk_wdt.h, 303
- hsk_wdt_enable
 - hsk_wdt.c, 301
 - hsk_wdt.h, 303
- hsk_wdt_init
 - hsk_wdt.c, 301
 - hsk_wdt.h, 303
- hsk_wdt_service
 - hsk_wdt.c, 301
 - hsk_wdt.h, 304
- I/O Port Pull-Up/-Down Setup, 64
 - IO_PORT_PULL_DISABLE, 64
 - IO_PORT_PULL_DOWN, 64
 - IO_PORT_PULL_ENABLE, 64
 - IO_PORT_PULL_INIT, 64
 - IO_PORT_PULL_UP, 65
- ICM7228_FACTORY
 - hsk_icm7228.h, 212
- IERR
 - hsk_isr8_callback, 85
- ILLUMINATE_OFFSET
 - hsk_icm7228.c, 208
- IO_PORT_DRAIN_DISABLE
 - Output Port Access, 61
- IO_PORT_DRAIN_ENABLE
 - Output Port Access, 61
- IO_PORT_GET
 - Input Port Access, 59
- IO_PORT_IN_INIT
 - Input Port Access, 60
- IO_PORT_ON_GND
 - Input Port Access, 60
- IO_PORT_ON_HIGH
 - Input Port Access, 60
- IO_PORT_OUT_INIT
 - Output Port Access, 61
- IO_PORT_OUT_SET
 - Output Port Access, 62
- IO_PORT_PULL_DISABLE
 - I/O Port Pull-Up/-Down Setup, 64
- IO_PORT_PULL_DOWN
 - I/O Port Pull-Up/-Down Setup, 64
- IO_PORT_PULL_ENABLE
 - I/O Port Pull-Up/-Down Setup, 64
- IO_PORT_PULL_INIT
 - I/O Port Pull-Up/-Down Setup, 64
- IO_PORT_PULL_UP
 - I/O Port Pull-Up/-Down Setup, 65
- IO_PORT_STRENGTH_STRONG
 - Output Port Access, 63
- IO_PORT_STRENGTH_WEAK
 - Output Port Access, 63
- IO_VAR_GET
 - Variable Access, 66
- IO_VAR_SET
 - Variable Access, 66
- IRDY
 - hsk_isr8_callback, 85
- ISR_hsk_isr14
 - hsk_isr.c, 224

- ISR_hsk_isr5
 - hsk_isr.c, [225](#)
- ISR_hsk_isr6
 - hsk_isr.c, [225](#)
- ISR_hsk_isr8
 - hsk_isr.c, [225](#)
- ISR_hsk_isr9
 - hsk_isr.c, [226](#)
- ISR_hsk_ssc
 - hsk_ssc.c, [283](#)
- ISR_hsk_timer0
 - hsk_timer01.c, [295](#)
- ISR_hsk_timer1
 - hsk_timer01.c, [295](#)
- ident
 - hsk_flash.c, [200](#)
- inBit
 - hsk_pwc.c, [247](#)
- inCount
 - hsk_pwc.c, [247](#)
- inSel
 - hsk_pwc.c, [247](#)
- init
 - main.c, [306](#)
- initialised
 - hsk_can.c, [156](#)
- Input Port Access, [59](#)
 - IO_PORT_GET, [59](#)
 - IO_PORT_IN_INIT, [60](#)
 - IO_PORT_ON_GND, [60](#)
 - IO_PORT_ON_HIGH, [60](#)
- invalid
 - hsk_pwc.c, [247](#)
- LEN_DFLASH
 - hsk_flash.c, [195](#)
- LEN_PFLASH
 - hsk_flash.c, [195](#)
- LEN_ROM
 - hsk_flash.c, [195](#)
- LEN_XRAM
 - hsk_flash.c, [195](#)
- LIST_NODEx
 - hsk_can.c, [133](#)
- LIST_PENDING
 - hsk_can.c, [133](#)
- LIST_UNALLOC
 - hsk_can.c, [134](#)
- lastCapture
 - hsk_pwc.c, [247](#)
- latest
 - hsk_flash.c, [201](#)
- MMC_DEFAULT
 - hsk_can.c, [134](#)
- MMC_GATEWAYSRC
 - hsk_can.c, [134](#)
- MMC_RXBASEFIFO
 - hsk_can.c, [134](#)
- MMC_TXBASEFIFO
 - hsk_can.c, [134](#)
- MMC_TXSLAVEFIFO
 - hsk_can.c, [134](#)
- MOAMRn
 - hsk_can.c, [134](#)
- MOARn
 - hsk_can.c, [134](#)
- MOCTRn
 - hsk_can.c, [135](#)
- MOD_MSEL6n
 - hsk_pwm.c, [264](#)
- MODATAHn
 - hsk_can.c, [135](#)
- MODATALn
 - hsk_can.c, [135](#)
- MOFCRn
 - hsk_can.c, [135](#)
- MOFGPRn
 - hsk_can.c, [135](#)
- MOFGPRn_BOT
 - hsk_can.c, [135](#)
- MOFGPRn_CUR
 - hsk_can.c, [135](#)
- MOFGPRn_SEL
 - hsk_can.c, [135](#)
- MOFGPRn_TOP
 - hsk_can.c, [136](#)
- MOSTATn
 - hsk_can.c, [136](#)
- MOSTATn_PNEXT
 - hsk_can.c, [136](#)
- MOVCI
 - hsk_flash.c, [195](#)
- main
 - main.c, [307](#)
- main.c, [304](#)
 - adc7, [313](#)
 - init, [306](#)
 - main, [307](#)
 - p1_buffer, [313](#)
 - p1_illuminate, [308](#)
 - p1_init, [310](#)
 - p1_refresh, [310](#)
 - p1_writeDec, [310](#)
 - p1_writeHex, [311](#)
 - p1_writeString, [311](#)
 - PERSIST_VERSION, [306](#)
 - persist, [313](#)
 - run, [311](#)
 - tick0, [312](#)
 - tick0_count_20, [313](#)
 - tick0_count_250, [313](#)
- modpiseIBit
 - hsk_ex.c, [180](#)
- modpiseISel
 - hsk_ex.c, [181](#)
- NBTRx

- hsk_can.c, 136
- NCRx
 - hsk_can.c, 136
- NDOV
 - hsk_isr5_callback, 81
 - hsk_isr8_callback, 85
- NECNTx
 - hsk_can.c, 136
- NFCRx
 - hsk_can.c, 136
- NIPRx
 - hsk_can.c, 137
- NMIECC
 - hsk_isr14_callback, 78
- NMIFLASH
 - hsk_isr14_callback, 78
- NMIPLL
 - hsk_isr14_callback, 78
- NMIVDDP
 - hsk_isr14_callback, 78
- NMIWDT
 - hsk_isr14_callback, 78
- NPCRx
 - hsk_can.c, 137
- NSRx
 - hsk_can.c, 137
- ndiv
 - hsk_boot.c, 115
- nextChannel
 - hsk_adc.c, 101
- nmidummy
 - hsk_isr.c, 226
- OFF_LISTm
 - hsk_can.c, 137
- OFF_MOn
 - hsk_can.c, 137
- OFF_MSIDk
 - hsk_can.c, 137
- OFF_MSPNDk
 - hsk_can.c, 137
- OFF_NODEx
 - hsk_can.c, 137
- oldest
 - hsk_flash.c, 201
- Output Port Access, 61
 - IO_PORT_DRAIN_DISABLE, 61
 - IO_PORT_DRAIN_ENABLE, 61
 - IO_PORT_OUT_INIT, 61
 - IO_PORT_OUT_SET, 62
 - IO_PORT_STRENGTH_STRONG, 63
 - IO_PORT_STRENGTH_WEAK, 63
- overflow
 - hsk_pwc.c, 248
 - hsk_timer01.c, 295
- overflows
 - hsk_pwc.c, 248
- p1_buffer
 - main.c, 313
- p1_illuminate
 - main.c, 308
- p1_init
 - main.c, 310
- p1_refresh
 - main.c, 310
- p1_writeDec
 - main.c, 310
- p1_writeHex
 - main.c, 311
- p1_writeString
 - main.c, 311
- PAGE_FLASH
 - hsk_flash.c, 195
- PAGE_RAM
 - hsk_flash.c, 195
- PAN_CMD_ALLOCBEFORE
 - hsk_can.c, 138
- PAN_CMD_ALLOCBEHIND
 - hsk_can.c, 138
- PAN_CMD_ALLOC
 - hsk_can.c, 138
- PAN_CMD_INIT
 - hsk_can.c, 138
- PAN_CMD_MOVEBEFORE
 - hsk_can.c, 138
- PAN_CMD_MOVEBEHIND
 - hsk_can.c, 138
- PAN_CMD_MOVE
 - hsk_can.c, 138
- PAN_CMD_NOP
 - hsk_can.c, 138
- PANAR1
 - hsk_can.c, 139
- PANAR2
 - hsk_can.c, 139
- PANCMD
 - hsk_can.c, 139
- PANCTR_READY
 - hsk_can.c, 139
- PANCTR
 - hsk_can.c, 139
- PANSTATUS
 - hsk_can.c, 139
- PDATA_PAGE
 - hsk_boot.c, 113
- PERSIST_VERSION
 - main.c, 306
- PRI_ID
 - hsk_can.c, 140
- PRI_LIST
 - hsk_can.c, 140
- PWC_CC0
 - hsk_pwc.h, 252
- PWC_CC0_EXINT_BIT
 - hsk_pwc.c, 236
- PWC_CC0_EXINT_REG

- hsk_pwc.c, [236](#)
- PWC_CC0_P30
 - hsk_pwc.h, [252](#)
- PWC_CC0_P40
 - hsk_pwc.h, [252](#)
- PWC_CC0_P55
 - hsk_pwc.h, [252](#)
- PWC_CC1
 - hsk_pwc.h, [253](#)
- PWC_CC1_EXINT_BIT
 - hsk_pwc.c, [236](#)
- PWC_CC1_EXINT_REG
 - hsk_pwc.c, [236](#)
- PWC_CC1_P32
 - hsk_pwc.h, [253](#)
- PWC_CC1_P41
 - hsk_pwc.h, [253](#)
- PWC_CC1_P56
 - hsk_pwc.h, [253](#)
- PWC_CC2
 - hsk_pwc.h, [253](#)
- PWC_CC2_EXINT_BIT
 - hsk_pwc.c, [236](#)
- PWC_CC2_EXINT_REG
 - hsk_pwc.c, [236](#)
- PWC_CC2_P33
 - hsk_pwc.h, [253](#)
- PWC_CC2_P44
 - hsk_pwc.h, [253](#)
- PWC_CC2_P52
 - hsk_pwc.h, [253](#)
- PWC_CC3
 - hsk_pwc.h, [254](#)
- PWC_CC3_EXINT_BIT
 - hsk_pwc.c, [237](#)
- PWC_CC3_EXINT_REG
 - hsk_pwc.c, [237](#)
- PWC_CC3_P34
 - hsk_pwc.h, [254](#)
- PWC_CC3_P45
 - hsk_pwc.h, [254](#)
- PWC_CC3_P57
 - hsk_pwc.h, [254](#)
- PWC_CHANNELS
 - hsk_pwc.c, [237](#)
- PWC_EDGE_BOTH
 - hsk_pwc.h, [254](#)
- PWC_EDGE_FALLING
 - hsk_pwc.h, [254](#)
- PWC_EDGE_RISING
 - hsk_pwc.h, [254](#)
- PWC_MODE_EXT
 - hsk_pwc.h, [254](#)
- PWC_MODE_SOFT
 - hsk_pwc.h, [255](#)
- PWC_UNIT_DUTYH_MS
 - Pulse Duty Times, [71](#)
- PWC_UNIT_DUTYH_NS
 - Pulse Duty Times, [71](#)
- PWC_UNIT_DUTYH_RAW
 - Pulse Duty Times, [71](#)
- PWC_UNIT_DUTYH_US
 - Pulse Duty Times, [72](#)
- PWC_UNIT_DUTYL_MS
 - Pulse Duty Times, [72](#)
- PWC_UNIT_DUTYL_NS
 - Pulse Duty Times, [72](#)
- PWC_UNIT_DUTYL_RAW
 - Pulse Duty Times, [72](#)
- PWC_UNIT_DUTYL_US
 - Pulse Duty Times, [72](#)
- PWC_UNIT_FREQ_H
 - Pulse Frequencies, [70](#)
- PWC_UNIT_FREQ_M
 - Pulse Frequencies, [70](#)
- PWC_UNIT_FREQ_S
 - Pulse Frequencies, [70](#)
- PWC_UNIT_SUM_RAW
 - Pulse Width Detection Units, [68](#)
- PWC_UNIT_WIDTH_MS
 - Pulse Width Times, [69](#)
- PWC_UNIT_WIDTH_NS
 - Pulse Width Times, [69](#)
- PWC_UNIT_WIDTH_RAW
 - Pulse Width Times, [69](#)
- PWC_UNIT_WIDTH_US
 - Pulse Width Times, [69](#)
- PWM_60
 - hsk_pwm.h, [271](#)
- PWM_61
 - hsk_pwm.h, [271](#)
- PWM_62
 - hsk_pwm.h, [271](#)
- PWM_63
 - hsk_pwm.h, [271](#)
- PWM_CC60
 - hsk_pwm.h, [271](#)
- PWM_CC61
 - hsk_pwm.h, [271](#)
- PWM_CC62
 - hsk_pwm.h, [272](#)
- PWM_COUT60
 - hsk_pwm.h, [272](#)
- PWM_COUT61
 - hsk_pwm.h, [272](#)
- PWM_COUT62
 - hsk_pwm.h, [272](#)
- PWM_COUT63
 - hsk_pwm.h, [272](#)
- PWM_OUT_60_P30
 - hsk_pwm.h, [272](#)
- PWM_OUT_60_P31
 - hsk_pwm.h, [272](#)
- PWM_OUT_60_P40
 - hsk_pwm.h, [272](#)
- PWM_OUT_60_P41

- hsk_pwm.h, [273](#)
- PWM_OUT_61_P00
 - hsk_pwm.h, [273](#)
- PWM_OUT_61_P01
 - hsk_pwm.h, [273](#)
- PWM_OUT_61_P31
 - hsk_pwm.h, [273](#)
- PWM_OUT_61_P32
 - hsk_pwm.h, [273](#)
- PWM_OUT_61_P33
 - hsk_pwm.h, [273](#)
- PWM_OUT_61_P44
 - hsk_pwm.h, [273](#)
- PWM_OUT_61_P45
 - hsk_pwm.h, [273](#)
- PWM_OUT_62_P04
 - hsk_pwm.h, [274](#)
- PWM_OUT_62_P05
 - hsk_pwm.h, [274](#)
- PWM_OUT_62_P34
 - hsk_pwm.h, [274](#)
- PWM_OUT_62_P35
 - hsk_pwm.h, [274](#)
- PWM_OUT_62_P46
 - hsk_pwm.h, [274](#)
- PWM_OUT_62_P47
 - hsk_pwm.h, [274](#)
- PWM_OUT_63_P03
 - hsk_pwm.h, [274](#)
- PWM_OUT_63_P37
 - hsk_pwm.h, [274](#)
- PWM_OUT_63_P43
 - hsk_pwm.h, [275](#)
- pdiv
 - hsk_boot.c, [115](#)
- persist
 - main.c, [313](#)
- portAltsel
 - hsk_ex.c, [181](#)
- portBit
 - hsk_ex.c, [181](#)
 - hsk_pwc.c, [248](#)
- portSel
 - hsk_pwc.c, [248](#)
- ports
 - hsk_ex.c, [181](#)
 - hsk_pwm.c, [267](#)
- pos
 - hsk_pwc.c, [248](#)
 - hsk_pwm.c, [267](#)
- prescaler
 - hsk_pwc.c, [248](#)
- ptr
 - hsk_flash.c, [201](#)
- ptr10
 - hsk_adc.c, [101](#)
- ptr8
 - hsk_adc.c, [101](#)

- Pulse Duty Times, [71](#)
 - PWC_UNIT_DUTYH_MS, [71](#)
 - PWC_UNIT_DUTYH_NS, [71](#)
 - PWC_UNIT_DUTYH_RAW, [71](#)
 - PWC_UNIT_DUTYH_US, [72](#)
 - PWC_UNIT_DUTYL_MS, [72](#)
 - PWC_UNIT_DUTYL_NS, [72](#)
 - PWC_UNIT_DUTYL_RAW, [72](#)
 - PWC_UNIT_DUTYL_US, [72](#)
- Pulse Frequencies, [70](#)
 - PWC_UNIT_FREQ_H, [70](#)
 - PWC_UNIT_FREQ_M, [70](#)
 - PWC_UNIT_FREQ_S, [70](#)
- Pulse Width Detection Units, [68](#)
 - PWC_UNIT_SUM_RAW, [68](#)
- Pulse Width Times, [69](#)
 - PWC_UNIT_WIDTH_MS, [69](#)
 - PWC_UNIT_WIDTH_NS, [69](#)
 - PWC_UNIT_WIDTH_RAW, [69](#)
 - PWC_UNIT_WIDTH_US, [69](#)
- RESET_DATA
 - hsk_can.c, [140](#)
- RESET
 - hsk_can.c, [140](#)
- rcount
 - hsk_ssc.c, [284](#)
- reset
 - hsk_flash_struct, [76](#)
- RI
 - hsk_isr8_callback, [85](#)
- rptr
 - hsk_ssc.c, [284](#)
- run
 - main.c, [311](#)
- SET_DATA
 - hsk_can.c, [140](#)
- SET
 - hsk_can.c, [140](#)
- SSC I/O Ports, [73](#)
 - SSC_MRST_P05, [73](#)
 - SSC_MRST_P14, [74](#)
 - SSC_MRST_P15, [74](#)
 - SSC_MTSR_P04, [74](#)
 - SSC_MTSR_P13, [74](#)
 - SSC_MTSR_P14, [74](#)
 - SSC_SCLK_P03, [74](#)
 - SSC_SCLK_P12, [74](#)
 - SSC_SCLK_P13, [74](#)
- SSC_BAUD
 - hsk_ssc.h, [287](#)
- SSC_CONF
 - hsk_ssc.h, [288](#)
- SSC_MASTER
 - hsk_ssc.h, [288](#)
- SSC_MRST_P05
 - SSC I/O Ports, [73](#)
- SSC_MRST_P14

- SSC I/O Ports, [74](#)
- SSC_MRST_P15
 - SSC I/O Ports, [74](#)
- SSC_MTSR_P04
 - SSC I/O Ports, [74](#)
- SSC_MTSR_P13
 - SSC I/O Ports, [74](#)
- SSC_MTSR_P14
 - SSC I/O Ports, [74](#)
- SSC_SCLK_P03
 - SSC I/O Ports, [74](#)
- SSC_SCLK_P12
 - SSC I/O Ports, [74](#)
- SSC_SCLK_P13
 - SSC I/O Ports, [74](#)
- SSC_SLAVE
 - [hsk_ssc.h](#), [288](#)
- STATE_DELETE
 - [hsk_flash.c](#), [195](#)
- STATE_DETECT
 - [hsk_flash.c](#), [196](#)
- STATE_IDLE
 - [hsk_flash.c](#), [196](#)
- STATE_REQUEST
 - [hsk_flash.c](#), [196](#)
- STATE_RESET
 - [hsk_flash.c](#), [196](#)
- STATE_WRITE
 - [hsk_flash.c](#), [196](#)
- sel
 - [hsk_pwm.c](#), [268](#)
- size
 - [hsk_flash.c](#), [201](#)
- state
 - [hsk_flash.c](#), [201](#)
 - [hsk_pwc.c](#), [248](#)
- sum
 - [hsk_pwc.c](#), [249](#)
- TF2
 - [hsk_isr5_callback](#), [81](#)
 - [hsk_isr8_callback](#), [85](#)
- targets
 - [hsk_adc.c](#), [101](#)
- TI
 - [hsk_isr8_callback](#), [85](#)
- tick0
 - [main.c](#), [312](#)
- tick0_count_20
 - [main.c](#), [313](#)
- tick0_count_250
 - [main.c](#), [313](#)
- timers
 - [hsk_timer01.c](#), [295](#)
- VAR_ASM
 - [hsk_flash.c](#), [196](#)
- VAR_AT
 - [hsk_flash.c](#), [196](#)
- Variable Access, [66](#)
 - [IO_VAR_GET](#), [66](#)
 - [IO_VAR_SET](#), [66](#)
- wcount
 - [hsk_ssc.c](#), [284](#)
- wptr
 - [hsk_ssc.c](#), [284](#)
- wrap
 - [hsk_flash.c](#), [201](#)
- XC878_16FF
 - [hsk_flash.h](#), [206](#)
- XRAM_BANK
 - [hsk_boot.c](#), [113](#)
- XRAM_SELECTOR
 - [hsk_boot.c](#), [113](#)
- xdataDptr
 - [hsk_flash.c](#), [201](#)