iC-TW8 16-BIT SIN/COS INTERPOLATOR
Programmer's Reference

CONTENTS

REGISTER MAP

Block	Register/Variable	Address	Bit	Type	Value Range	Pin. Cfg. Val.	Name, Description, Comments	Ref.
LUT	**LUT Block**	**0x8000 - 0x8041**					**Look-Up Table Data**	PR7
	LUT0...LUT63	0x8000 - 0803F		b	-127...+127		**LUT Data**	PR7
	LUT_CKSUM	0x8040		w (16 bit)			**LUT Checksum**	PR7
CFG	**CFG Block**	**0x8042 - 0x8067**					**Configuration Data**	**PR7**
General	CFG_CKSUM	0x8042 0x8043		w (16 bit)			**CFG Block Checksum** NB: automatically calculated when writing to the EEPROM.	PR13
	MAIN_CFG.wp	0x8044	0		0, 1	wp = 1	**Pin WP - EEPROM Write Lock** 0: pin WP high after startup, 1: pin WP low after startup	PR14
	MAIN_CFG.pull	0x8044	2		0, 1	pull = 0	**Input Pull-Up/Pull-Down Resistors** 0: disabled, 1: enabled	PR14
	MAIN_CFG.clkout	0x8044	3		0, 1		**Signal Output to Pin CLOCK and Pin FRAME** 0: disabled, 1: enabled Pin CLOCK: internal Delta-Sigma Modulator Clock Pin FRAME: internal ADC Sampling Clock	PR14
	MAIN_DSM.freq	0x8045	2...0		0 - 6		**PWM Output Frequency** 0: fdsm = fcore / 4, 1: fdsm = fcore / 8... 6: fdsm = fcore / 256 7: reserved, do not use.	PR14
	fdsm				kHz		**Delta-Sigma Modulator Clock Frequency** *fdsm = fcore / 2e(MAIN_DSM.freq + 2)*	PR14
	MAIN_DSM.clamp	0x8045	3		0, 1		**PWM Generator Clamping** 0: unclamped (PWM wraps around), 1: clamped	PR14
	fadc				kHz		**ADC Sampling Clock Frequency** *fadc = fcore / 128*	PR14
Ipo	MAIN_INTER	0x8046 0x8047	7...0 7...0	w (16 bit)	0, 4 - 65535	set by C0, C1	**AB Output Resolution in Edges** 0: 65536 (interpolation factor x16384) 1 - 3: reserved, do not use. 4 - 65535 *(inter: x1 - x16383.75)*	PR15
	inter						**Interpolation Factor** *inter = MAIN_INTER / 4*	PR15
	MAIN_HYST	0x8048 0x8049	7...0 1	w (9 bit)	0 - 511	set by C2	**Signal Path Output Hysteresis** 0: min, 511: max (+/- 22°) **Eval.board:** 1 LSB, 2 LSB, 4 LSB, 8 LSB (C2 heavy/normal/off)	PR15
	hyst						**Input Referred Angle Hysteresis** *hyst = MAIN_HYST x 360 / 8192*	PR15

Block	Register/Variable	Address	Bit	Type	Value Range	Pin. Cfg. Val.	Name, Description, Comments	Ref.
Filter	MAIN_FLTR.fb	0x804A	0	b	0, 1	set by C3	**Output Filter Feedback Path Delay** 0: no delay in feedback path (lag recovery disabled) 1: delay in feedback path (lag recovery enabled) **Eval.board:** C3 disabled = 0, enabled = 1	PR15
	delay						**Output Filter Feedback Path Delay** *delay = 640 / fcore*	PR15
	MAIN_FLTR.p	0x804A	3...1	b	0 - 6	set by C2	**Output Filter Proportional Gain** 0: pgain= 1, 1: pgain= 0.5, ... 6: pgain= 0.015625 7: reserved, do not use. **Eval.board:** Latency recovery disabled (by C3), pgain= 0.25 (C2 heavy), 0.5 (normal), 1.0 (off) Latency recovery enabled (by C3), pgain = 0.031 (C2 heavy), 0.063 (normal), 0.125 (off)	PR15
	pgain						**Output Filter Proportional Gain** *pgain = 1 / 2e MAIN_FLTR.p*	PR15
	MAIN_FLTR.i	0x804A	6...4	b	0 - 7	set by C2	**Output Filter Integral Gain** 0: igain= 1, 1: igain= 0.5, ... 6: igain= 0.015625 7: igain= 0 **Eval.board:** Latency recovery disabled (by C3), igain= 0 (C2 heavy/normal/off) Latency recovery enabled (by C3), igain= 0.016 (C2 heavy), 0.031 (normal), 0.063 (off)	PR16
	igain						**Output Filter Integral Gain** *igain = 1 / 2e MAIN_FLTR.i*	PR16
	MAIN_OUT.start	0x804B	3...0		0 - 15	start = 10	**Power-Up Wait Time for Analog Circuitry Settling** 10: twait[ms] = 1048 / fcore[MHz]: 33 ms @ 32 MHz	PC7,14 PR16
	twait						**Actual Startup Delay** *twait = 128 / fcore x (2e(MAIN_OUT.start + 3) — 1)*	PR16
	MAIN_OUT.mode	0x804B	5...4		0 - 2	mode = 0	**Operating Mode** 0: AB output mode, 1: serial-only output mode 2: PWM output mode, 3: reserved, do not use.	PR16
Clock	MAIN_CLOCK.xtal	0x804C	0		0, 1	xtal = 1	**Clock Source** 0: internal osc, 1: ext. crystal or ext. oscillator	PR17
	MAIN_CLOCK.div	0x804C	2...1		0, 2, 3	div = 3	**Internal Oscillator Clock Divider** 0: internal oscillator /4, 1: reserved (do not use) 2: internal oscillator /2, 3: internal oscillator	PR17
	MAIN_CLOCK.freq	0x804C	6...3		0 - 15		**Internal Oscillator Frequency Tuning** 0: min, 15: max	PR17
	fxtl				El.Char. 401		**Crystal Frequency**	
	fosc				El.Char. 402		**Internal Oscillator Frequency**	
	fcore						**Clock Frequency of Main Signal Path** (i.e. tuned internal freq. - or ext. crystal freq.)	
	MAIN_CLOCK.xforce	0x804C	7		0, 1	xforce = 1	**Clock Source on Low Power** 0: internal osc, 1: use always ext. crystal or oscillator Must be 1 for proper operation of chip release D3.	PR17
	Reserved	0x804D	7...0	byte	0x00		Must be 0x00 for proper operation.	PR13
	Internal Clock Reg.						Same function as MAIN_CLOCK, but immediate impact. An update command is not required.	
	RB_CLOCK.xtal	0xA004	0		0, 1		**Clock Source** 0: internal osc, 1: ext. crystal or ext. oscillator	
	RB_CLOCK.div	0xA004	2...1		0, 2, 3		**Internal Oscillator Clock Divider** 0: internal oscillator /4, 1: reserved (do not use) 2: internal oscillator /2, 3: internal oscillator	
	RB_CLOCK.freq	0xA004	6...3		0 - 15		**Internal Oscillator Frequency Tuning** 0: min, 15: max	
	Reserved	0xA004	7		0, 1		Must be 1 for proper operation.	
	Reserved	0xA005	7...0	byte	0x00		Must be 0x00 for proper operation.	

Block	Register/Variable	Address	Bit	Type	Value Range	Pin. Cfg. Val.	Name, Description, Comments	Ref.
Index	MAIN_ZPOS	0x804E 0x804F	7...0 5...0	w (14 bit)	0 - 16383		**Z Output Index Position** 0: 0°, ... , 8192: 180°, ... , 16384: 359.98° NB: when gated by the zero inputs.	SC27, PR17
	zpos						**Actual Z Index Position** *zpos = MAIN_ZPOS x 360 / 16384*	SC27, PR17
	MAIN_Z.th	0x8050	4...0	4-bit 2K	+/- 15		**Zero Input Switching Threshold** -15: -450 mV, 0: 0 mV, +15: +450 mV	PR18
	zth						**Zero Input Switching Threshold** *zth = MAIN_Z.th x 30 mV*	PR18
	MAIN_Z.reset	0x8050	5	b	0, 1	reset = 0	**32-bit Position Counter Reset Configuration** 0: counter reset at zpos, 1: no counter reset	PR18
	Reserved	0x8050	7...6	b	0, 1		Bits must be 0 for proper operation.	PR13
	Reserved	0x8051	7...0	byte	0x00		Must be 0x00 for proper operation.	PR13
AB	AB_CFG0.dir	0x8052	0	b	0, 1	dir = 0	**AB Counting Direction** 0: Normal counting direction 1: Reversed counting direction	PR18
	AB_CFG0.apol	0x8052	1	b	0, 1	apol = 0	**A Output Polarity / State at Z Position*** 0: Normal A polarity and A output low at ZPOS 1: Inverted A polarity and A output high at ZPOS NB: *) for AB_CFG0.startmode = 1 or 3, AB_CFG1.div = 0, and an integer value for *inter*	PR18
	AB_CFG0.bpol	0x8052	2	b	0, 1	bpol = 0	**B Output Polarity / State at Z Position*** 0: Normal B polarity and A output low at ZPOS 1: Inverted B polarity and A output high at ZPOS NB: *) for AB_CFG0.startmode = 1 or 3, AB_CFG1.div = 0, and an integer value for *inter*	PR18
	AB_CFG0.zpol	0x8052	3	b	0, 1	zpol = 0	**Z Output Polarity** 0: Z is active high, 1: Z is active low	PR18
	AB_CFG0.startmode	0x8052	6...5	b	0, 1, 3	startmode = 1	**ABZ Output Startup Behavior** 0: relative, 1: same phase, 3: burst 2: reserved, do not use.	PC14, PR19
	AB_CFG1.div	0x8053	4...0	b	0 - 31		**Post-AB Divider** 0: post-AB division disabled 1 - 31: post-AB division	PR19
	div						**Actual Post-AB Division** *div = AB_CFG1.div + 1* *NB: inter ≥ div required*	PR19
	intereff						**Effective Interpolation Factor** *intereff = inter / div*	PR19
	AB_ZWIDTH	0x8054	7...0	byte	0 - 255		**Z Output Pulse Width** 0: min, 255: max NB: AB_ZWIDTH must be less than MAIN_INTER.	PR19
	zwidth						**Actual Z Output Pulse Width** *zwidth = (AB_ZWIDTH + 1) / div*	PR19
	AB_VTOP	0x8055	6...0		0 - 127	set by C3	**AB Output Frequency Limit** 0: max, 127: min	PR20
	fab						**Actual AB Output Frequency Limit** *fab = fcore / (4 x (AB_VTOP + 1) x div)*	PR20
	Reserved	0x8056	7...0	byte	0x00		Must be 0x00 for proper operation.	PR13
	Reserved	0x8057	7...0	byte	0x00		Must be 0x00 for proper operation.	PR13

Block	Register/Variable	Address	Bit	Type	Value Range	Pin. Cfg. Val.	Name, Description, Comments	Ref.
Adapt	**ADPT_CFG**					set by C1	**Auto-Adaption** 0: disabled, 1: enabled	
	ADPT_CFG.doff	0x8058	0		0, 1		Digital Offset	PR20
	ADPT_CFG.dgain	0x8058	1		0, 1		Digital Gain	PR20
	ADPT_CFG.dphase	0x8058	2		0, 1		Digital Phase	PR20
	ADPT_CFG.aoff	0x8058	3		0, 1		Analog Offset	PR20
	ADPT_CFG.again	0x8058	4		0, 1		Analog Gain	PR20
	ADPT_CFG.lut	0x8058	5		0, 1		Look-Up Table	PR20
	ADPT_CFG.store	0x8058	6		0, 1		Auto-Storage to EEPROM	PR20
	ADPT_DETAIL.flimit	0x8059	2...0		0 - 7	flimit = 6	**Auto-Adaption Frequency Limit** 0: min, 7: max	PR21
	fadapt						**Actual Auto-Adaption Frequency Limit** *fadapt = fcore / 83886.08 x 2eADPT_DETAIL.flimit*	PR21
	ADPT_DETAIL.tbase	0x8059	5...3		0 - 7	tbase = 4	**Auto-Adaption Correction Time Base** 0: min, 7: max	PR21
	fbase						**Actual Auto-Adaption Time Base** *fbase = 2e(ADPT_DETAIL.tbase+7) / fcore*	PR21
	ADPT_DETAIL.fault	0x8059	6		0, 1		**Auto-Adaption During ADC Fault** 0: continued, 1: disabled	PR21
	ADPT_STORE.offth	0x805A	3...0		0 - 15		**Dig. Offs. Threshold for Auto-Storage to EEPROM** 0: min, 15: max	PR22
	ADPT_STORE.gainth	0x805A	7...4		0 - 15		**Dig. Gain Threshold for Auto-Storage to EEPROM** 0: min, 15: max	PR22
	ADPT_CORR.prop	0x805B	1...0		0 - 3	prop = 2	**Auto-Adaption Control** 0: linear correction *(1 increment / tbase)* 1: slow exponential *(prop = 0.25)* 2: medium exponential *(prop = 0.5)* 3: fast exponential *(prop = 0.75)*	PR22
	ADPT_CORR.offtol	0x805B	4...2		0 - 7	offtol = 2	**Auto-Adaption Dig. Offset Tolerance** 0: min, 7: max	PR22
	ADPT_CORR.gaintol	0x805B	7...5		0 - 7	gaintol = 2	**Auto-Adaption Dig. Gain Tolerance** 0: min, 7: max	PR22
	Reserved	0x805C	7...0	byte	0x00		Must be 0x00 for proper operation.	PR13

Block	Register/Variable	Address	Bit	Type	Value Range	Pin. Cfg. Val.	Name, Description, Comments	Ref.
Moni	**MON_CFG**						**ADC and Adaption Quality Monitoring**	
	MON_CFG.adc	0x805D	0		0, 1		**ADC Quality Monitoring Enable** 0: disabled, 1: enabled	PR23
	MON_CFG.adapt	0x805D	1		0, 1		**Adaption Quality Monitoring Enable** 0: disabled, 1: enabled NB: Must be enabled if excessive adaption should be recorded by STAT_SP.adapt.	PR23
	MON_CFG.pwm	0x805D	3...2		0 - 3		**STATUS Output PWM Control** 0: dutycycle *pwm* /255 1, 2, 3: refer to datasheet	PR23
	pwm						**PWM Signal** *pwm = QM_ADC x MON_CFG.adc + QM_ADAPT x MON_CFG.adapt*	PR23
	MON_CFG.adcth	0x805D	6		0, 1		ADC Quality Threshold (MSB of MON_ADC.th) 0: $0 \leq$ *adcth* < 120 1: $120 \leq$ *adcth* < 184	PR23
	MON_ADC.limit	0x805E	3...0		0 - 15		**ADC Underflow Fault Limit** 0: highest, 15: lowest limit *adclim = 2e(0.5x MON_ADC.limit + 1)*	PR24
	MON_ADC.th	0x805E	7...4		0 - 15		**ADC Quality Threshold** 0: lowest, 15: highest threshold *adcth = 4x (MON_ADC.th + 16x MON_CFG.adcth) + 60*	PR24
	adcflt						**Actual ADC Underflow Fault** *adcflt = adcth - adclim*	PR24
	MON_OFF.limit	0x805F	3...0		0 - 12 0 - 7	limit = 10	**Dig. Offset Adaption Limit** 0: lowest, 12: highest limit (deviation limit allowed from base configuration) 0: lowest, 7: highest limit if monitoring function is enabled (only for chip releases before D4) 13-15: reserved, do not use. *offlim = 2e(MON_OFF.limit)*	PR24, PR35
	MON_OFF.th	0x805F	7...4		0 - 12 0 - 7		**Dig. Offset Adaption Quality Threshold** 0: lowest, 12: highest threshold 0: lowest, 7: highest limit if monitoring function is enabled (only for chip releases before D4) 13-15: reserved, do not use. *offqth = 2e(MON_OFF.th)*	PR24, PR35
	offflt				0 - 8192		**Max. Digital Offset Adaption and Fault Level** *offflt = offqth + offlim*	PR24, PR35
	MON_GAIN.limit	0x8060	3...0		0 - 12 0 - 7	limit = 10	**Dig. Gain Match Adaption Limit** 0: lowest, 12: highest limit (deviation limit allowed from base configuration) 0: lowest, 7: highest limit if monitoring function is enabled (only for chip releases before D4) 13-15: reserved, do not use. *gainlim = 2e(MON_GAIN.limit)*	PR24, PR35
	MON_GAIN.th	0x8060	7...4		0 - 12 0 - 7		**Dig. Gain Match Adaption Quality Threshold** 0: lowest, 12: highest threshold 0: lowest, 7: highest limit if monitoring function is enabled (only for chip releases before D4) 13-15: reserved, do not use. *gainqth = 2e(MON_GAIN.th)*	PR24, PR35
	gainflt				0 - 8192		**Max. Digital Gain Match Adaption and Fault Level** *gainflt = gainqth + gainlim*	PR24, PR35
	MON_PHASE.limit	0x8061	3...0		0 - 12 0 - 7	limit = 10	**Dig. Phase Adaption Limit** 0: lowest, 12: highest limit (deviation limit allowed from base configuration) 0: lowest, 7: highest limit if monitoring function is enabled (only for chip releases before D4) 13-15: reserved, do not use. *phlim = 2e(MON_PHASE.limit)*	PR24, PR35
	MON_PHASE.th	0x8061	7...4		0 - 12 0 - 7		**Dig. Phase Adaption Quality Threshold** 0: lowest, 12: highest threshold 0: lowest, 7: highest limit if monitoring function is enabled (only for chip releases before D4) 13-15: reserved, do not use. *phqth = 2e(MON_PHASE.th)*	PR24, PR35
	phflt				0 - 8192		**Max. Phase Adaption and Fault Level** *phflt = phqth + phlim*	PR24, PR35

Block	Register/Variable	Address	Bit	Type	Value Range	Pin. Cfg. Val.	Name, Description, Comments	Ref.
Fault	**FLT_CFG**						**Fault Configuration**	PR26
	FLT_CFG.vwarn	0x8062	0	b	0, 1	vwarn = 1	**Pin FAULT Indication of Operational Warnings** 0= ignored, 1= activates FAULT output NB: 1 indicates overspeed and flickering index signal.	PR26
	FLT_CFG.hold	0x8062	1, 2	b	0 - 3		**AB Output Hold** 0: AB outputs operational during a fault 1, 2: reserved (do not use) 3: AB outputs disabled during a fault	PR26
	FLT_CFG.pol	0x8062	3	b	0, 1		**Pin FAULT Polarity** 0: active low, 1: active high	PR26
	FLT_CFG.long	0x8062	4	b	0, 1	long = 1	**Pin FAULT Indication Time** 0: active while fault is active, 1: activation prolonged	PR27
	long						**Actual Pin FAULT Indication Time** *long[ms] = 20 x 32/fcore[MHz]*	PR27
	Reserved	0x8063	7...0	byte	0x00		Must be 0x00 for proper operation.	PR13
	FLT_STAT						**Fault Status Register** 0: no fault, 1: fault (sticky, cleared by writing 0x00) NB: this register is not stored in the EEPROM.	PR27
	FLT_STAT.ee	0x8064	0	b	0, 1		EEPROM Fault (by STAT_EE)	PR27
	FLT_STAT.adc	0x8064	1	b	0, 1		ADC Fault (by STAT_SP: .adcof, .adcuf)	PR27
	FLT_STAT.adapt	0x8064	2	b	0, 1		Adaption Fault (by STAT_SP.adapt)	PR27
	FLT_STAT.vwarn	0x8064	3	b	0, 1		Overspeed Warning (by STAT_SP.vtop) NB: warning comes at half the fatal overspeed fault.	SC18, PR28
	FLT_STAT.vfatal	0x8064	4	b	0, 1		Fatal Overspeed Fault (by STAT_SP.fatal)	PR28
	FLT_STAT.intern	0x8064	6	b	0, 1		Internal Fault (by STAT_SP.1wire and .xtal)	PR28
	Reserved	0x8065	7...0	byte	0x00		Must be 0x00 for proper operation.	PR13
	FLT_EN						**Fault Latching Enable**	PR28
	FLT_EN.ee	0x8066	0	b	0, 1	ee = 1	EEPROM Fault 0: disabled, 1: latches FAULT output	PR28
	FLT_EN.adc	0x8066	1	b	0, 1		ADC Fault 0: activates FAULT output, 1: latches FAULT output NB: defines STAT_SP.adcof and .adcuf as sticky; can freeze auto-adaption upon pwr-on.	PR28
	FLT_EN.adapt	0x8066	2	b	0, 1		Adaption Fault 0: disabled (recommended) 1: latches FAULT output, freezes auto-adaption	PR28, SC23
	FLT_EN.vwarn	0x8066	3	b	0, 1		Overspeed Warning (Operational Warning) 0: activates FAULT outp. (only if FLT_CFG.vwarn = 1) 1: latches FAULT output NB: defines STAT_SP.vtop and .ospeed as sticky (not recommended)	PR28
	FLT_EN.vfatal	0x8066	4	b	0, 1	vfatal = 1	Fatal Overspeed Fault 0: disabled, 1: latches FAULT output NB: defines STAT_SP.fatal, .lag and .ab as sticky (recommended)	PR29
	FLT_EN.intern	0x8066	6	b	0, 1		Internal Fault 0: activates FAULT output, 1: latches FAULT output	PR29
	Reserved	0x8067	7...0	byte	0x00		Must be 0x00 for proper operation.	PR13

Block	Register/Variable	Address	Bit	Type	Value Range	Pin. Cfg. Val.	Name, Description, Comments	Ref.
VAR	VAR Block	0x8094 - 0x9FFF					Working Variables (internal RAM)	PR7
Status	**STAT_SP**	0x809A		w (16 bit)			**Signal Path Status Register** 0: no fault, 1: fault (sticky, cleared by writing 0x00) (For some variables: FLT_EN determines if sticky or not)	PR32
	STAT_SP.adcof	0x809A	0		0, 1		ADC Overflow NB: ADC input > ≈ 106 % nom. signal level (±1.5 V).	PR33
	STAT_SP.adcuf	0x809A	1		0, 1		ADC Underflow NB: ADC input < ≈ 10 % nominal signal level. Threshold changes with ADC quality monitoring enabled.	PR33
	STAT_SP.adapt	0x809A	2		0, 1		Excessive Adaption (operational if enabled by MON_CFG.adapt=1) NB: always sticky; can only be cleared by restarting the chip to read new parameter values from the EEPROM	PR33, SC23
	STAT_SP.vtop	0x809A	3		0, 1		Speed Limit Exceeded (max. AB frequency reached)	PR33
	STAT_SP.ospeed	0x809A	4		0, 1		Overspeed Warning (operation continues correctly)	PR33
	STAT_SP.fatal	0x809A	5		0, 1		Fatal Overspeed Fault (faulty AB output signals) NB: pin FAULT latching is highly recommended (faulty interpolator output)	SC23, PR33
	STAT_SP.lag	0x809A	6		0, 1		Excessive Filter Lag (faulty AB output signals)	PR33
	STAT_SP.ab	0x809A	7		0, 1		Excessive AB Output Lag (faulty AB output signals)	PR33
	Reserved	0x809A	8		0, 1		reserved	PR33
	STAT_SP.1wire	0x809A	9		0, 1		1-Wire Interface Timeout (access/cmd. not performed) NB: always sticky	PR33
	Reserved	0x809A	10		0, 1		reserved	PR32
	STAT_SP.xtal	0x809A	11		0, 1		External Crystal Fault (internal oscillator in use) NB: cleared when frequency/crystal is applied.	PR33
	Reserved	0x809A	12...15		0, 1		reserved	PR33
	STAT_EE	0x809C		w (16 bit)			**EEPROM Status Register** 0: no fault, 1: fault (sticky, cleared by writing 0x00)	PR35
	STAT_EE.timeout	0x809C	0		0, 1		EEPROM Communication Timeout	PR35
	STAT_EE.stuck	0x809C	1		0, 1		EEPROM Hardware Fault	PR35
	STAT_EE.id	0x809C	2		0, 1		Wrong EEPROM ID	PR35
	STAT_EE.cfg	0x809C	3		0, 1		CFG Block Checksum Error	PR35
	STAT_EE.par0	0x809C	4		0, 1		PAR Block 0 Checksum Error	PR35
	STAT_EE.par1	0x809C	5		0, 1		PAR Block 1 Checksum Error	PR35
	STAT_EE.base	0x809C	6		0, 1		PAR_BASE Block Checksum Error	PR35
	STAT_EE.lut	0x809C	7		0, 1		LUT Block Checksum Error	PR35
	STAT_EE.mode	0x809C	8		0, 1		Wrong Configuration Mode	PR35
	Reserved	0x809C	9...15		0, 1		reserved	PR35
QM	ADC_SIN	0x80A4	15...0	w (16 bit)	-8192... +8192		**Sine ADC Value** NB: recommended maximum ±5800 (for nominal signal level of ±1.4 V).	PR35
	ADC_COS	0x80A6	15...0	w (16 bit)	-8192... +8192		**Cosine ADC Value** NB: recommended maximum ±5800 (for nominal signal level of ±1.4 V.	PR35
	ADC_AMP	0x80A8	7...0	byte	0 - 255		**Vector Signal Amplitude**	PR36
	QM_ADC	0x80AE	7...0	byte	0 - 255		**ADC Quality Monitor Level**	PR36
	QM_ADAPT	0x80AF	7...0	byte	0 - 255		**Adaption Quality Monitor Level**	PR36
CMD	CMD	0x80B0	7...0	byte	selected		**Command Register** 0x00: executed - command register ready (read only) 0x01: reserved (do not use) 0x02: update signal path configuration NB: Update clears test register RB_TEST1. 0x04: write all (to EEPROM) 0x08: write param (block PAR_NOW to EEPROM) 0x10: reserved (do not use) 0x20: reserved (do not use) 0x40: restart (as if power had been cycled) 0x80: reserved (do not use) NB: Check for 0x00 before writing new command. If using the 1-wire interface, wait for 500 ms upon restart instead of polling for 0x00.	PR33

Block	Register/Variable	Address	Bit	Type	Value Range	Pin. Cfg. Val.	Name, Description, Comments	Ref.
RB	**RB Block**	**0xA000 - 0xA0FF**					**Register Banks (internal RAM)**	PR9
Test	RB_TEST1	0xA008		w (16 bit)			**Test Register** 0x0000: Cleared 0x0008 (bit 3 = 1): Output of osc. frequency fcore/2 at FRAME. 0x0020 (bit 5 = 1): Output of internal Z Gating Signal at OUTA. 0x0200 (bit 9 = 1): EEPROM unlock. NB: Clear register or use update command before writing to the EEPROM.	PR37
WM	**WM Block**	**0xA100 - 0xFFFF**					**Working Memory Block (internal RAM, ROM)**	PR7
	CHIP_ID	0xC002		w (16 bit)	0x0408		iC-TW8 Device ID (read only)	PR9
	CHIP_REV	0xC004		w (16 bit)	$\geq$ 0x0001		iC-TW8 Chip Release (read only) NB: chip release D3 = 0x0003, D4 = 0x0004.	PR9
	POS_LSW	0xA18E		w (16 bit)	0 - 0xFFFF		Current Angle Position (32-bit)	PR9
	POS_MSW	0xA190		w (16 bit)	0 - 0xFFFF		Current Angle Position (32-bit)	PR9
	VEL	0xA192		w (13 bit)	+/- 0x1FFF		Current Velocity (Input Frequency) and Sense of Direction	PR9
	finput				kHz		**Actual Input Frequency** *finput[kHz] = fcore[MHz] x VEL/2097.152*	PR9
	finput_max				kHz		**Max. Input Frequency (calc. as ffront by REDT)** *finput_max [kHz] = fab / intereff*	—
PAR	**PAR Block**	**0x806A - 0x8093**					**Signal Path Parameters**	PR7
	PAR_NOW Block	0x806A - 0x8077					**Current Parameter Values**	PR8
	PAR_CKSUM	0x806A		w (16 bit)			Checksum PAR_NOW Sub-Block	
	PAR_DGAIN	0x806C		w (16 bit)	-1023..+1023		**Digital Gain Match Correction** -1023...+1023: digital gain correction -1024..-32767, +1024..+32767: reserved, do not use.	PR31
	dgains						**Actual Digital Gain Match Correction Sin** *DGAIN < 0: dgains = 1 - DGAIN/4096, dgainc = 1;*	PR31
	dgainc						**Actual Digital Gain Match Correction Cos** *DGAIN > 0: dgainc = 1 + DGAIN/4096, dgains = 1;*	PR31
	PAR_DOFFS	0x806E		w (16 bit)	-511...+511		**Digital Sin Offset Correction** -511...+511: digital offset correction -512...-32767, +512...+32767: reserved, do not use.	PR31
	doffs						**Actual Digital Sin Offset Correction** *doffs[mV] = DOFFS/4096*	PR31
	PAR_DOFFC	0x8070		w (16 bit)	-511...+511		**Digital Cos Offset Correction** -511...+511: digital offset correction -512...-32767, +512...+32767: reserved, do not use.	PR31
	doffc						**Actual Digital Cos Offset Correction** *doffc[mV] = DOFFC/4096*	PR31
	PAR_DPH	0x8072		w (16 bit)	-1023..+1023		**Digital Phase Correction** -1023...+1023: digital phase correction -1024..-32767, +1024..+32767: reserved, do not use.	PR32
	dphase				-53°...+53°		**Actual Digital Phase Correction** *dphase[°] = 2x arctan (DPH/2048)*	PR31
	PAR_AGAIN	0x8074		byte	2 - 15		**Analog Gain** 0 - 1: reserved, do not use. 2: minimum, 15: maximum 16 - 255: reserved, do not use.	PR30
	again				2 - 178 6 dB - 45 dB		**Actual Analog Gain (for Sin and Cos)** *again = 10e (3x AGAIN / 20)* *again[dB] = 3x AGAIN*	PR30
	PAR_AOFFS	0x8075		byte	-31...+31		**Analog Sin Offset Correction** -31...+31: analog offset correction -32...-127, +32...+127: reserved, do not use.	PR30
	aoffs						**Actual Analog Sin Offset Correction** *aoffs[mV] = AOFFS * 100*	PR30
	aoffins						**Effective Analog Sin Input Offset Correction** *aoffins[mV] = aoffs / again*	PR30
	PAR_AOFFC	0x8076		byte	-31...+31		**Analog Cos Offset Correction** -31...+31: analog offset correction -32...-127, +32...+127: reserved, do not use.	
	aoffc						**Actual Analog Cos Offset Correction** *aoffc[mV] = AOFFC * 100*	PR30
	aoffinc						**Effective Analog Cos Input Offset Correction** *aoffinc[mV] = aoffc / again*	PR30

Block	Register/Variable	Address Bit	Type	Value Range	Pin. Cfg. Val.	Name, Description, Comments	Ref.
	PAR_BAK Block	0x8078 - 0x8085				**Backup Parameter Values**	PR8
	PAR_BAK_CKSUM	0x8078	w (16 bit)			Checksum PAR_BAK_Sub-Block	
	PAR_BAK_DGAIN	0x807A	w (16 bit)			Digital Gain	
	PAR_BAK_DOFFS	0x807C	w (16 bit)			Sin Digital Offset	
	PAR_BAK_DOFFC	0x807E	w (16 bit)			Cos Digital Offset	
	PAR_BAK_DPH	0x8080	w (16 bit)			Digital Phase	
	PAR_BAK_AGAIN	0x8082	byte			Analog Gain	
	PAR_BAK_AOFFS	0x8083	byte			Sin Analog Offset	
	PAR_BAK_AOFFC	0x8084	byte			Cos Analog Offset	
	PAR_BASE Block	0x8086 - 0x8093				**Base Parameter Values**	PR8
	PAR_BASE_CKSUM	0x8086	w (16 bit)			Checksum PAR_BASE_Sub-Block	
	PAR_BASE_DGAIN	0x8088	w (16 bit)			Digital Gain	
	PAR_BASE_DOFFS	0x808A	w (16 bit)			Sin Digital Offset	
	PAR_BASE_DOFFC	0x808C	w (16 bit)			Cos Digital Offset	
	PAR_BASE_DPH	0x808E	w (16 bit)			Digital Phase	
	PAR_BASE_AGAIN	0x8090	byte			Analog Gain	
	PAR_BASE_AOFFS	0x8091	byte			Sin Analog Offset	
	PAR_BASE_AOFFC	0x8092	byte			Cos Analog Offset	

CONFIGURATION PROCEDURE

Serial Configuration Procedure for SPI and 1-Wire Interface

1. The complete CFG block should be written first. All addresses starting at **0x8042 through 0x8067**. Undescribed bits completing a byte or word must be programmed zero.
2. Then the PAR_NOW sub-block starting at **0x806C through 0x8076**.
3. Then a Write All Command (0x04 to 0x80B0) to load the other PAR sub-blocks and calculate and store the checksums.
4. Finally, a Restart command (0x40 to 0x80B0) to read the EEPROM back into internal memory and update the signal path parameters.

REVISION HISTORY

Rel.	Rel. Date*	Chapter	Modification	Page
B3	2017-08-09	REGISTER MAP	LUT block moved from page 7 to 1 MAIN_Z.th: fullscale values and zth formula corrected finput_max: description corrected (ffront by REDT) STAT_SP.adcof, STAT_SP.adcuf: notes added ADC_SIN, ADC_COS: note text updated	1ff
		REVISION HISTORY	Chapter added	9
		ATTACHMENTS	Programmer's Reference updated (Aug 09, 2017)	PR1...39

Rel.	Rel. Date*	Chapter	Modification	Page
B4	2017-10-25	REGISTER MAP	Description updated for AB_CFG0.dir, AB_CFG0.apol, AB_CFG0.bpol	3
		ATTACHMENTS	Programmer's Reference updated (Oct 19, 2017)	PR1...39

* Release Date format: *YYYY-MM-DD*

iC-TW8 16-BIT SIN/COS INTERPOLATOR
Programmer's Reference

Table of Contents

About This Document

This document describes the memory configuration, internal registers, and variables of the iC-TW8 Interpolator. It is meant to be used as a reference for interpreting values read back from internal registers during troubleshooting or for writing diagnostic or configuration software for a microcontroller or computer connected to the iC-TW8 via the SPI or 1-wire communication interfaces. This document should not be used to configure the iC-TW8; separate documents describing both pin configuration mode and serial configuration mode using the iC-TW8 encoder design tool are available for this purpose.

Documentation Conventions

Throughout this document, hexadecimal values are written in C-style where the hex value is preceded by "0x". For example, $80CA_{16}$ is written as 0x80CA.

Internal iC-TW8 registers are named using all CAPITAL letters. Individual bits or bit groups within the register use lower case Roman lettering. For example, MAIN_CFG.wp refers to the wp bit (bit 0) in the MAIN_CFG register at address 0x8044. Likewise, MAIN_OUT.mode refers to the 2-bit group mode (bits 4 – 5) of register MAIN_OUT at address 0x804B.

Derived variables not explicitly available as register values are shown in all lower-case *italics*. For example, the formula

$$inter = \frac{\mathrm{MAIN\_INTER}}{4}$$

shows that internal variable *inter* is derived from the value in the MAIN_INTER register.

Internal variable units are shown in brackets following the variable name. For instance *adcs*[V] is the sin channel ADC input level in volts.

Serial Communication Ports

The iC-TW8 contains two serial ports that may be used to access internal registers. These are a standard SPI (Serial Peripheral Interface) slave port and a 1-wire port, both of which utilize a similar communication protocol. Both ports may be used simultaneously.

The SPI slave port uses a 3-wire or 4-wire full-duplex interface which operates in CPOL = 0 and CPHA = 0 mode only. This means that the base (resting) value of SPI_SCLK (pin 27) is low, SPI_SI is sampled on the rising edge of SPI_SCLK, and SPI_SO is changed on the falling edge of SPI_CLCK.

The SPI port does not have a time-out circuit to check for valid transactions. If a transaction is aborted, the internal state of the port registers is maintained. To reset the port, the SPI_xSS input (pin 14) may be pulled high or sufficient clock cycles provided to flush the port registers back to their initial state.

The 1-wire port uses a single bi-directional data line connected to IR (pin 7) for communication. The bit stream is pulse-width modulated: a 1-bit is encoded as a long high level followed by a short low level, a 0-bit is encoded as a short high level followed by a long low level. A timeout circuit activates a status bit in the STAT_SP register (page 33) in the event of an illegal transaction.

SPI Port

The SPI interface allows three types of transactions (read, write, and 32-bit position read) as shown in Figure 1 below. All SPI commands, addresses, data, and position values are read and written most significant bit first.

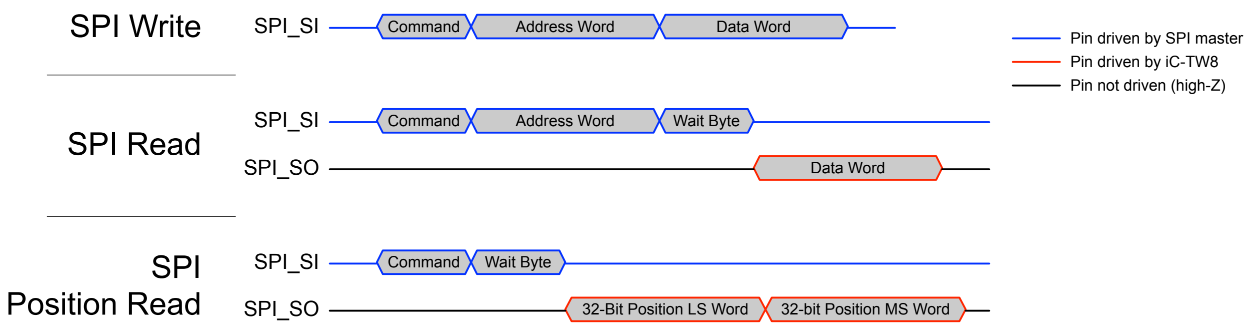

Figure 1: SPI Transactions

An SPI write consists of a command byte followed by a 16-bit register address and then 16 bits of data. For a word write, the data word is written to the register address, which must be even. For a byte write, the least significant byte of the data word is written to the register address; the most significant byte of the data word is ignored but must be provided.

An SPI read consists of a command byte followed by a 16-bit address and an 8-bit "dummy" waiting time during which the iC-TW8 retrieves the requested data from internal memory. Subsequently, the 16-bit data is provided on SPI_SO. For a word read, 16 bits are read from the register address, which must be even. For a byte read, 8 bits are read from the register address and returned in the least significant byte of the data word; the most significant byte of the data word is undefined.

An SPI position read is similar to an SPI read except that an address is not required and 32 bits of data are returned in two words. The resolution of the returned position is determined by the interpolation factor, *inter*. See MAIN_INTER on page 15 for more information.

The position is latched on SPI clock 6 and updated at the ADC sampling clock frequency, *fadc*.

$$fadc = \frac{fcore}{128}$$

This position update rate is asynchronous to any SPI accesses, so care must be taken when making continuous position reads to avoid jitter and aliasing. See Using Serial-Only Output Mode in the iC-TW8 Serial Configuration Mode documentation for more information.

The Command Byte determines whether the transaction is a read or a write, a word or a byte, or a position read. Five command byte values are legal; other values cause undefined operation.

SPI Command Byte	
Value	**Description**
0x80	Word Write
0x90	Byte Write
0xC0	Word Read
0xD0	Byte Read
0xE0	32-Bit Position Read

An SPI Write is detailed in Figure 2 below.

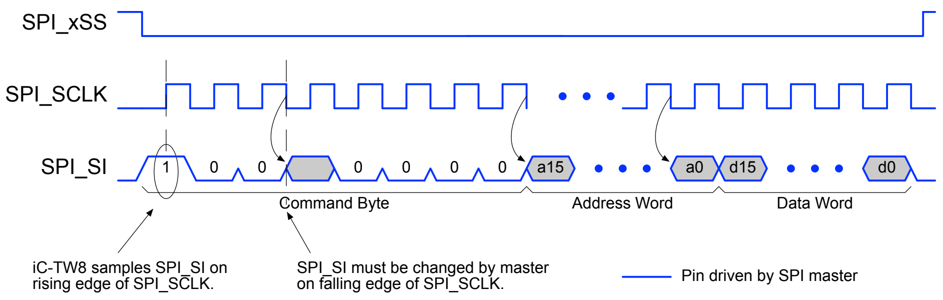
SPI_xSS
SPI_SCLK
SPI_SI
1
0
0
0
0
0
0
a15
a0
d15
d0
Command Byte
Address Word
Data Word
iC-TW8 samples SPI_SI on rising edge of SPI_SCLK.
SPI_SI must be changed by master on falling edge of SPI_SCLK.
Pin driven by SPI master

Figure 2: SPI Write

An SPI Read is detailed in Figure 3 below.

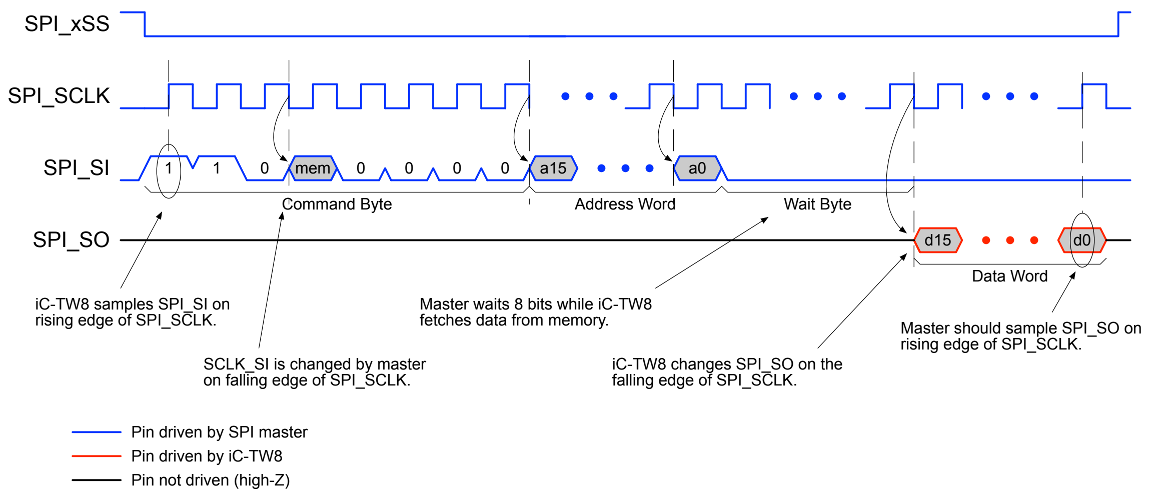
SPI_xSS
SPI_SCLK
SPI_SI
SPI_SO
1
1
0
mem
0
0
0
0
a15
a0
d15
d0
Command Byte
Address Word
Wait Byte
Data Word
iC-TW8 samples SPI_SI on rising edge of SPI_SCLK.
Master waits 8 bits while iC-TW8 fetches data from memory.
Master should sample SPI_SO on rising edge of SPI_SCLK.
SCLK_SI is changed by master on falling edge of SPI_SCLK.
iC-TW8 changes SPI_SO on the falling edge of SPI_SCLK.
Pin driven by SPI master
Pin driven by iC-TW8
Pin not driven (high-Z)

Figure 3: SPI Read

An SPI 32-Bit Position Read is detailed in Figure 4 below.

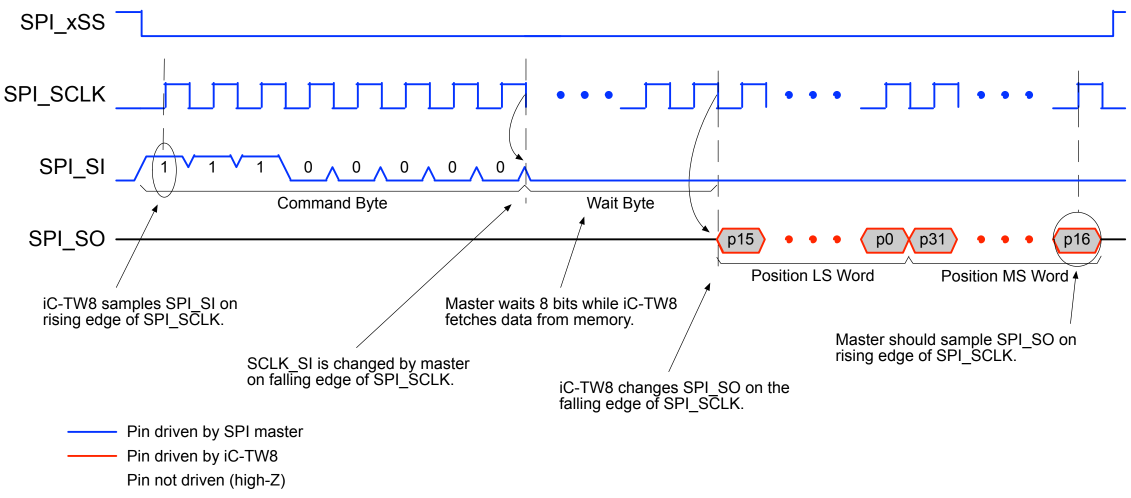
SPI_xSS
SPI_SCLK
SPI_SI
SPI_SO
1
1
1
0
0
0
0
0
Command Byte
Wait Byte
p15
p0
p31
p16
Position LS Word
Position MS Word
iC-TW8 samples SPI_SI on rising edge of SPI_SCLK.
Master waits 8 bits while iC-TW8 fetches data from memory.
SCLK_SI is changed by master on falling edge of SPI_SCLK.
iC-TW8 changes SPI_SO on the falling edge of SPI_SCLK.
Master should sample SPI_SO on rising edge of SPI_SCLK.
Pin driven by SPI master
Pin driven by iC-TW8
Pin not driven (high-Z)

Figure 4: SPI 32-Bit Position Read

Note that the 32-bit position value is simply an accumulated count of the number of AB increments since the iC-TW8 was started or restarted. Thus, it is not a true multicycle counter except with binary interpolation factors (where MAIN_INTER is a power of two).

For example, if MAIN_INTER = 4096, the lowest 12 bits of the 32-bit position value represent the position or angle within one input cycle while the 20 MSBs represent the cycle count. However, if MAIN_INTER = 4000, no such straightforward representation is possible.

With non-binary resolutions, the angle within one input cycle and the cycle count must be calculated in a host processor using the 32-bit position value. In this case, it is recommended that the host processor maintain angle and multicycle count registers that are updated periodically using the difference between the current and the last 32-bit position values. This allows the rollover of the 32-bit position counter in the iC-TW8 to be handled properly.

1-Wire Port

The 1-wire port is intended for use as an alternative configuration interface and not for general communication with the iC-TW8. It allows two types of transactions (read and write) and uses a communication protocol similar to that used by the SPI port.

A 1-wire write consists of a start condition followed by the command byte followed by a 16-bit register address and then 16 bits of data. For a word write, the data word is written to the register address, which must be even. For a byte write, the least significant byte of the data word is written to the register address; the most significant byte of the data word is ignored but must be provided.

A 1-wire read consists of a start condition followed by the command byte followed by a 16-bit address, after which the master must release the data line. The iC-TW8 then pulls the data line low while retrieving the requested data from internal memory. Subsequently, the 16-bit data is provided on the data line. For a word read, 16 bits are read from the register address, which must be even. For a byte read, 8 bits are read from the register address and returned in the least significant byte of the data word; the most significant byte of the data word is undefined.

The Command Byte determines whether the transaction is a read or a write, or a word or a byte. Four command byte values are legal; other values cause undefined operation.

1-Wire Command Byte	
Value	**Description**
0x80	Word Write
0xA0	Byte Write
0xC0	Word Read
0xE0	Byte Read

A 1-wire write is detailed in Figure 5 below.

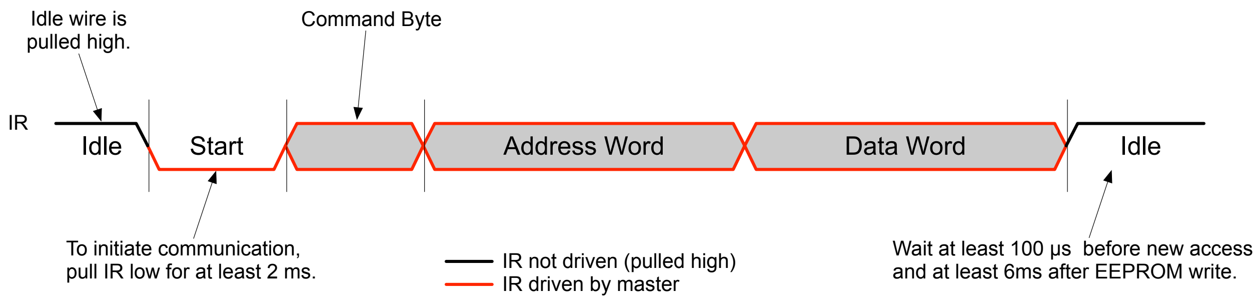
Idle wire is pulled high.
Command Byte
IR
Idle
Start
Address Word
Data Word
Idle
To initiate communication, pull IR low for at least 2 ms.
IR not driven (pulled high)
IR driven by master
Wait at least 100 µs before new access and at least 6ms after EEPROM write.

Figure 5: 1-Wire Write

A 1-wire read is detailed in Figure 6 below.

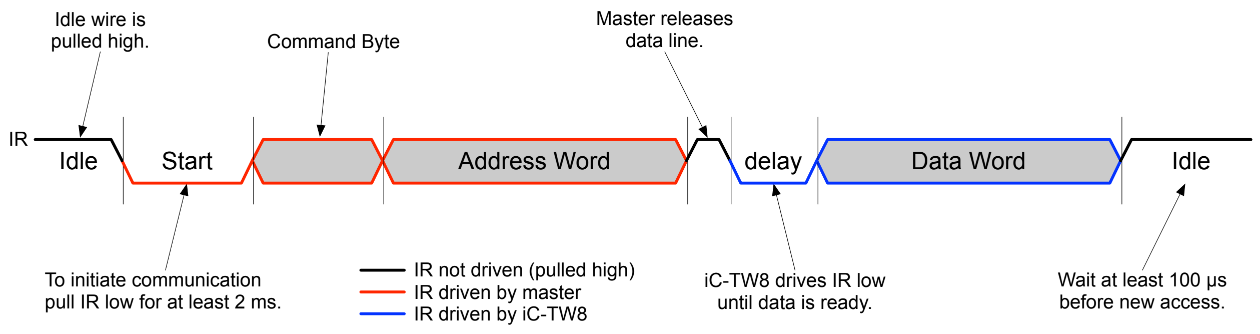
Idle wire is pulled high.
Command Byte
Master releases data line.
IR
Idle
Start
Address Word
delay
Data Word
Idle
To initiate communication pull IR low for at least 2 ms.
IR not driven (pulled high)
IR driven by master
IR driven by iC-TW8
iC-TW8 drives IR low until data is ready.
Wait at least 100 µs before new access.

Figure 6: 1-Wire Read

Memory Structure

The iC-TW8 contains two distinct memory spaces, the internal memory and the external EEPROM. Internal memory can be fully accessed through the serial SPI or 1-wire interfaces.

The external EEPROM is used to permanently store configuration and calibration data. All interactions with the external EEPROM are handled by the iC-TW8. At startup the iC-TW8 accesses the EEPROM and loads the configuration data into its internal memory. Several data integrity checks are performed to ensure proper configuration.

Internal Memory

The internal memory of the iC-TW8 consists of 32K bytes that can be accessed by the user via the SPI/1-wire interface. Memory is usually accessed as words (2 bytes = 16 bits) at an even byte address, although some locations also allow access as bytes (8 bits).

The internal memory of the iC-TW8 is divided into functional blocks, as shown in the memory map below.

Address Range	Block Name	Description
0x0000 – 07FFF	NU	Not Used
0x8000 – 0x8041	LUT	Lookup Table Data
0x8042 – 0x8067	CFG	Configuration Data
0x8068 – 0x8093	PAR	Signal Path Parameters
0x8094 – 0x9FFF	VAR	Working Variables
0xA000 – 0xA0FF	RB	Register Banks
0xA100 – 0xFFFF	WM	Working Memory

NU Block

The NU block of memory is not used by the iC-TW8. Accesses to this block have no effect.

LUT Block

The LUT block holds the values of the lookup table used to correct for sensor distortion. The LUT consists of 64 bytes of data plus a checksum word as shown below.

Address Range	Name	Description
0x8000 – 0x803F	LUT0 – LUT63	LUT Data
0x8040	LUT_CKSUM	Checksum

If the lookup table is disabled, values in this memory block are not used. LUT data should be accessed as bytes, but the LUT checksum must be accessed as a word.

CFG Block

The CFG block contains configuration data that controls the operational state of the iC-TW8. The CFG block can be accessed as bytes or words as required by the specific data. Word accesses must use an even byte address. See CFG Block Registers on page 13 for details.

PAR Block

The PAR block contains the signal path parameters for gain, offset correction, and phase correction used during operation of the interpolator. The PAR block can be accessed as bytes or words as required by the specific data. Word accesses must use an even byte address. See PAR Block Registers on page 30 for details.

The PAR block also contains a pointer variable PAR_ACTIVE at address 0x8068 that determines which of two EEPROM parameter blocks is used at power-up. See External Memory (EEPROM) on page 10 for more information on how PAR_ACTIVE is used.

Because the signal path parameter values can be updated during operation using the auto-adaption feature of the iC-TW8, the PAR block keeps three copies of all parameter values in three identically-structured sub-blocks as shown below.

Address Range	Sub-Block Name	Description
0x806A – 0x8077	PAR_NOW	Current Values
0x8078 – 0x8085	PAR_BAK	Backup Values
0x8086 – 0x8093	PAR_BASE	Baseline Values

The PAR_NOW sub-block contains the current parameter values used by the signal path. These values are adapted during operation to compensate for changes in operating conditions if the iC-TW8 is configured to use auto-adaption.

The PAR_NOW sub-block is structured as shown below.

Address	Parameter	Description	Access
0x806A	CKSUM	PAR_NOW Sub-Block Checksum	Word
0x806C	DGAIN	Digital Gain	Word
0x806E	DOFFS	Sin Digital Offset	Word
0x8070	DOFFC	Cos Digital Offset	Word
0x8072	DPH	Digital Phase	Word
0x8074	AGAIN	Analog Gain	Byte
0x8075	AOFFS	Sin Analog Offset	Byte
0x8076	AOFFC	Cos Analog Offset	Byte

The PAR_BAK sub-block contains the parameter values at the time of the last power-on. Immediately after power-on, the values in PAR_BAK are equal to the corresponding values in PAR_NOW. If the iC-TW8 is configured *not* to use auto-adaption, the parameter values in PAR_BAK are *always* the same as the corresponding values in PAR_NOW.

The PAR_BAK sub-block is structured identically to the PAR_NOW sub-block as shown below.

Address	Parameter	Description	Access
0x8078	CKSUM	Checksum of PAR_BAK Sub-Block	Word
0x807A	DGAIN	Digital Gain	Word
0x807C	DOFFS	Sin Digital Offset	Word
0x807E	DOFFC	Cos Digital Offset	Word
0x8080	DPH	Digital Phase	Word
0x8082	AGAIN	Analog Gain	Byte
0x8083	AOFFS	Sin Analog Offset	Byte
0x8084	AOFFC	Cos Analog Offset	Byte

The PAR_BASE sub-block contains the baseline parameter values determined during product configuration. PAR_BASE always provides a known factory default set of operating parameter values that do not change even when using auto-adaption.

The PAR_BASE sub-block is structured identically to the PAR_NOW sub-block as shown below.

Address	Parameter	Description	Access
0x8086	CKSUM	PAR_BASE Sub-Block Checksum	Word
0x8088	DGAIN	Digital Gain	Word
0x808A	DOFFS	Sin Digital Offset	Word
0x808C	DOFFC	Cos Digital Offset	Word
0x808E	DPH	Digital Phase	Word
0x8090	AGAIN	Analog Gain	Byte
0x8091	AOFFS	Sin Analog Offset	Byte
0x8092	AOFFC	Cos Analog Offset	Byte

VAR Block

The VAR block contains the CMD (command) register, the signal path status register, the EEPROM status register, and the iC-TW8's working variable values as shown below.

Address	Variable	Description	Access
0x809A	STAT_SP	Signal Path Status	Word
0x809C	STAT_EE	EEPROM Status	Word
0x80A4	ADC_SIN	Sin ADC Value	Word
0x80A6	ADC_COS	Cos ADC Value	Word
0x80A8	ADC_AMP	Signal Amplitude	Byte
0x80AE	QM_ADC	ADC Quality Monitor	Byte
0x80AF	QM_ADAPT	Adaption Quality Monitor	Byte
0x80B0	CMD	Command Register	Byte

The variables shown above may be read from the VAR block for troubleshooting or test purposes, but writing to these locations may cause unpredictable results. See VAR Block Registers on page 33 for details.

RB Block

The RB block provides access to internal register banks of the iC-TW8. These register banks are used to invoke special test modes. The RB block must be accessed as words at an even byte address.

Address	Register	Description
0xA008	RB_TEST1	Test Register

The RB_TEST1 Register is used to unlock the EEPROM (see EEPROM Write Protection on page 12), measure the internal oscillator frequency, and test the index gating.

WM Block

The WM block is the working memory of the iC-TW8. Writing to any of these locations may cause unpredictable results.

There are two variables in the WM Block that may be read to identify the iC-TW8 and its revision level as shown below.

Address	Variable	Value	Access
0xC002	CHIP_ID	0x0408	Word
0xC004	CHIP_REV	≥ 0x0001	Word

The LSB of the CHIP_ID is always 08 to identify the iC-TW8, while the MSB is incremented if major functional changes are made. CHIP_REV is incremented for minor improvements that do not impact functionality.

Also available in the WM block are the current sensor position (angle) and velocity as shown below.

Address	Variable	Value	Access
0xA18E	POS_LSW	0 – 0xFFFF	Word
0xA190	POS_MSW	0 – 0xFFFF	Word
0xA192	VEL	±0x1FFF	Word

Sensor position is accumulated in an internal 32-bit counter that can be read as two 16-bit words. It is important to read POS_LSW first to latch the complete 32-bit counter value. After reading POS_LSB, POS_MSB can be read and the complete 32-bit position value assembled. The resolution of the 32-bit position is determined by the interpolation factor, *inter*. See MAIN_INTER on page 15 for more information.

The 32-bit position is updated at the ADC sampling clock frequency, *fadc*.

$$fadc = \frac{fcore}{128}$$

This position update rate is asynchronous to any SPI accesses, so care must be taken when making continuous position reads to avoid jitter and aliasing. See Using Serial-Only Output Mode in the iC-TW8 Serial Configuration Mode documentation for more information.

Instantaneous sensor input velocity (frequency) is also available as a signed 13-bit value in the variable VEL. The actual sensor input frequency, *finput*, is calculated as

$$finput[\text{kHz}] = \frac{\text{VEL}}{2097.152} \cdot fcore[\text{MHz}]$$

Note that the resolution of VEL is not affected by the interpolation factor.

External Memory (EEPROM)

All access to the external EEPROM is controlled by the iC-TW8. The EEPROM is only read at startup. Parameter values and configuration can be written to the EEPROM using commands sent to the iC-TW8 using the SPI/1-wire interface.

The EEPROM stores startup values for the PAR_NOW sub-block, the CFG block, and the LUT block. The EEPROM memory space is defined as shown below.

Name	Description
PAR_ACTIVE	Pointer to Active Startup Parameter Block
PAR_0 Block	Startup Parameters Block 0
PAR_1 Block	Startup Parameters Block 1
PAR_BASE block	Default Parameters Block
CFG Block	Startup CFG Block
LUT Block	Startup LUT Block

Because the signal path parameters in the PAR_NOW sub-block can be changed by auto-adaption and optionally stored back to the EEPROM during operation, a double buffer is implemented to ensure data integrity in case an EEPROM write cycle is interrupted. As shown above, the EEPROM stores two complete sets of startup parameters as the PAR_0 block and the PAR_1 block. Which of these two blocks will be used at startup is determined by the PAR_ACTIVE variable stored in the EEPROM.

Reading the EEPROM

The data stored in the EEPROM is read into the corresponding blocks of internal memory only at startup. The normal startup sequence for reading the EEPROM is shown below.

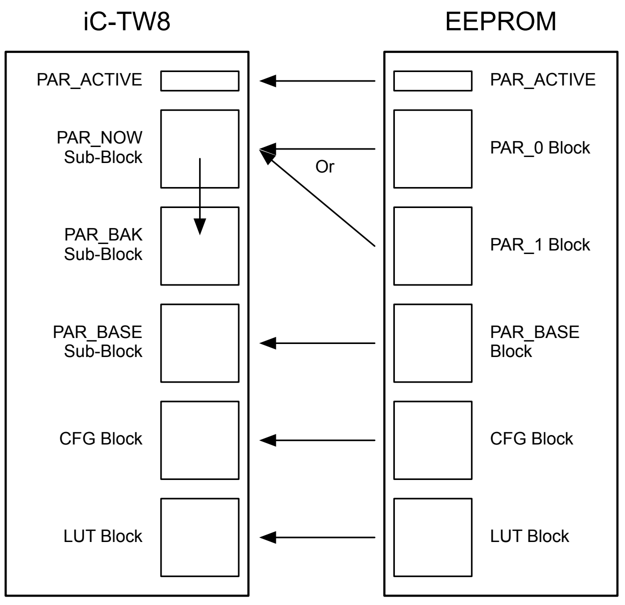

Figure 7: EEPROM Read

In a normal startup, the iC-TW8 reads the PAR_ACTIVE variable from the EEPROM to determine the active parameter set. If PAR_ACTIVE = 0, the PAR_0 block is copied from the EEPROM to the PAR_NOW sub-block in internal memory. If PAR_ACTIVE = 1, the PAR_1 block is copied from the EEPROM to the PAR_NOW sub-block in internal memory. Then the PAR_BASE, CFG, and LUT blocks are copied from the EEPROM to the corresponding blocks in internal memory. Finally, the iC-TW8 copies the PAR_NOW sub-block to the PAR_BAK sub-block.

An abnormal startup occurs if the iC-TW8 detects a checksum error in one of the EEPROM blocks. In this case, the appropriate error bit or bits in the STAT_EE register is set (See VAR Block Registers on page 33). If the active PAR block in the EEPROM has a checksum error, the other PAR block is selected. If it does not have a checksum error, that block is loaded into PAR_NOW and startup continues. If that block also has a checksum error, the PAR_BASE block is selected. If the PAR_BASE block does not have a checksum error, it is loaded into PAR_NOW and startup continues.

If the PAR_BASE block also has a checksum error, startup in aborted and the signal path is not enabled. This double-buffered EEPROM read assures that the iC-TW8 always starts up with a valid set of signal path parameters unless all three startup parameter blocks have checksum errors.

If the startup CFG block or the startup LUT block in the EEPROM has a checksum error, startup is also aborted and the signal path is not enabled.

Writing to the EEPROM

Two commands are available to write to the EEPROM. See CMD (Command Register) on page 33 for more information.

The Write All command copies complete configuration data from internal memory to the EEPROM as shown below.

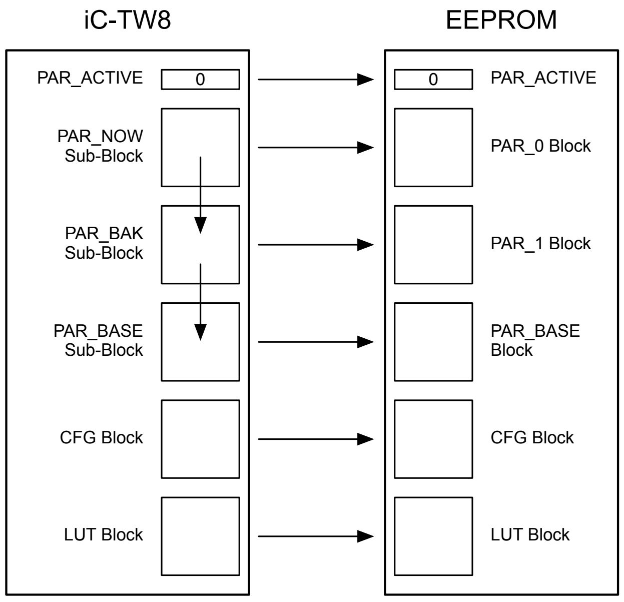

Figure 8: EEPROM Write All

When a Write All commend is executed, PAR_ACTIVE is reset to 0, the PAR_NOW sub-block is copied to the PAR_BAK and PAR_BASE sub-blocks, and then checksums for all three sub-blocks are calculated and stored. Checksums for the CFG and LUT blocks are also calculated and stored. Finally, PAR_ACTIVE and the complete PAR, CFG, and LUT blocks are copied from internal memory to the EEPROM.

The Write Parameters command copies the PAR_NOW sub-block from internal memory to the non-active parameter block in EEPROM as shown below.

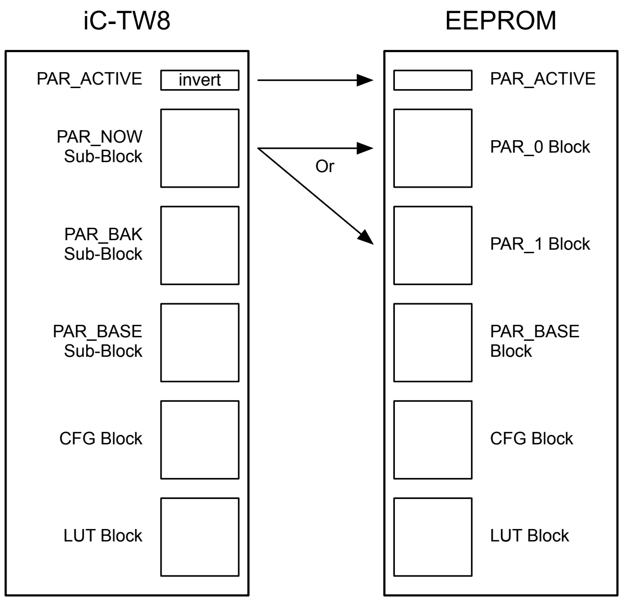

Figure 9: EEPROM Write Parameters

When a Write Parameters commend is executed, the checksum for the PAR_NOW sub-block is calculated and stored in internal memory. PAR_ACTIVE is inverted (PAR_ACTIVE = /PAR_ACTIVE) to point to the inactive startup PAR block in EEPROM. The PAR_NOW sub-block is then copied from internal memory to the currently inactive PAR block in the EEPROM. Finally, the new PAR_ACTIVE value is copied to the EEPROM, making the newly written parameter set active for the next startup. In this way the Write Parameters command always preserves the previous set of signal path parameters as a backup should there be an error writing the new set.

EEPROM Write Protection

The iC-TW8 can only successfully write data to an un-protected (unlocked) EEPROM. If the external EEPROM is write protected, the Write All and Write Parameters commands will fail with no indication (i.e. no fault or status bit is set). Thus it is imperative that the EEPROM be unlocked before attempting to write to it.

EEPROM write protection can be controlled by the iC-TW8 or by user-defined circuitry. If the iC-TW8 WP output (pin 29) is *not* connected to the EEPROM WP input, then the EEPROM must be manually unlocked using whatever means are provided by the external circuitry connected to the EEPROM.

If the iC-TW8 WP output (pin 29) is connected to the EEPROM WP input, then write protection of the EEPROM is controlled by the iC-TW8 and the EEPROM must be unlocked via the iC-TW8. To unlock the external EEPROM, write 0x0200 to RB_TEST1 at address 0xA008. To lock the external EEPROM, write 0x0000 to RB_TEST1. These actions over-ride the default EEPROM write protection stored in the CFG block (see MAIN_CFG on page 14).

CFG Block Registers

The CFG block contains configuration data that controls the operational state of the iC-TW8. During operation, the CFG block register values do not change, with the exception of the FLT_STAT register, which is updated with status information.

At startup, startup CFG block values are copied from the external EEPROM to the CFG block in internal memory (except FLT_STAT, which is not saved in the external EEPROM). The values in internal memory may be changed by the user during operation by writing to the appropriate register in the CFG block, but the startup values are restored from the EEPROM at the next startup.

The CFG block can be accessed as words or bytes depending on the specific register as shown below. Reserved registers must be set to 0x00 and unnamed bits must be set to 0 as shown below for proper operation. Any changes to CFG Block register values must be followed by an Update Command to take effect. See CMD (Command Register) on page 33.

Address	Register or Variable	Register Description or Variable Name								Access
		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
0x8042	CFG_CKSUM	CFG Block Checksum								Word
0x8044	MAIN_CFG	0	0	0	0	clkout	pull	0	wp	Byte
0x8045	MAIN_DSM					clamp	freq			Byte
0x8046	MAIN_INTER	Interpolation Factor (AB edges or increments per input cycle)								Word
0x8048	MAIN_HYST	Hysteresis								Word
0x804A	MAIN_FLTR	0	i			p			fb	Byte
0x804B	MAIN_OUT	0	0	mode		start				Byte
0x804C	MAIN_CLOCK	xforce	freq				div		xtal	Byte
0x804D	Reserved	Must be 0x00 for proper operation								Byte
0x804E	MAIN_ZPOS	Z Output Position								Word
0x8050	MAIN_Z	0	0	reset	th					Byte
0x8051	Reserved	Must be 0x00 for proper operation								Byte
0x8052	AB_CFG0	0	startmode		0	zpol	bpol	apol	dir	Byte
0x8053	AB_CFG1	0	0	0	div					Byte
0x8054	AB_ZWIDTH	Z Output Pulse Width								Byte
0x8055	AB_VTOP	0	AB Output Frequency Limit							Byte
0x8056	Reserved	Must be 0x00 for proper operation								Byte
0x8057	Reserved	Must be 0x00 for proper operation								Byte
0x8058	ADPT_CFG	0	store	lut	again	aoff	dphase	dgain	doff	Byte
0x8059	ADPT_DETAIL	0	fault	tbase			flimit			Byte
0x805A	ADPT_STORE	gainth				offth				Byte
0x805B	ADPT_CORR	gaintol			offtol			prop		Byte
0x805C	Reserved	Must be 0x00 for proper operation								Byte
0x805D	MON_CFG	0	adcth	0	0	pwm		adapt	adc	Byte
0x805E	MON_ADC	th				limit				Byte
0x805F	MON_OFF	th				limit				Byte
0x8060	MON_GAIN	th				limit				Byte
0x8061	MON_PHASE	th				limit				Byte
0x8062	FLT_CFG	0	0	0	long	pol	hold		vwarn	Byte
0x8063	Reserved	Must be 0x00 for proper operation								Byte
0x8064	FLT_STAT	0	intern	0	vfatal	vwarn	adapt	adc	ee	Byte
0x8065	Reserved	Must be 0x00 for proper operation								Byte
0x8066	FLT_EN	0	intern	0	vfatal	vwarn	adapt	adc	ee	Byte
0x8067	Reserved	Must be 0x00 for proper operation								Byte

CFG_CHECK

The CFG_CKSUM register holds the checksum for the CFG block. CFG_CKSUM is automatically calculated when the CFG block is written to EEPROM (see Writing to the EEPROM on page 2).

MAIN_CFG

The MAIN_CFG register contains bits that are used to activate various hardware features of the iC-TW8.

If the iC-TW8 WP (Write Protect) output (pin 29) is connected to the EEPROM WP input, then MAIN_CFG.wp determines whether the EEPROM is locked or unlocked during operation as shown below.

MAIN_CFG.wp (Byte 0x8044 Bit 0)	
Value	**Description**
0	EEPROM locked after startup (pin 29 high)
1	EEPROM unlocked after startup (pin 29 low)

The iC-TW8 can be configured to use pull-up and pull-down resistors on the sensor Sin/Cos inputs to enhance fault detection in the event of a floating sensor input.

MAIN_CFG.pull (Byte 0x8044 Bit 2)	
Value	**Description**
0	No pull-ups on sensor inputs
1	Pull-ups on sensor inputs

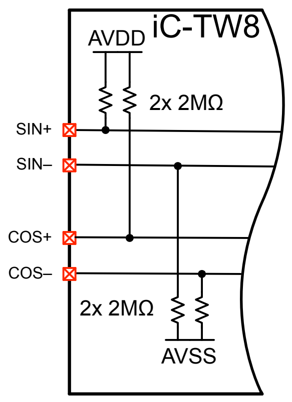

Figure 10: Input Pull-up Resistors

With MAIN_CFG.pull = 1, if a sensor input becomes disconnected, that input is immediately pulled to a level which will cause an ADC fault (see FLT_STAT on page 27).

MAIN_CFG.clkout determines whether the iC-TW8 CLOCK (pin 6) and FRAME (pin 8) outputs are active.

MAIN_CFG.clkout (Byte 0x8044 Bit 3)	
Value	**Description**
0	CLOCK and FRAME outputs are inactive
1	CLOCK and FRAME outputs are active

With MAIN_CFG.clkout = 1, the internal DSM (Delta-Sigma Modulator) clock is available on the CLOCK output (pin 6) and the ADC (Analog to Digital Converter) sampling clock is available on the FRAME output (pin 8). The internal DSM (Delta-Sigma Modulator) clock frequency, *fdsm*, is determined by MAIN_DSM.freq (see page 14). The ADC sampling clock frequency, *fadc*, is

$$fadc = \frac{fcore}{128}$$

MAIN_DSM

The MAIN_DSM register contains variables that configure the differential pulse-width modulated output in PWM output mode. If PWM output mode is *not* used, MAIN_DSM should be set to 0x00.

In PWM output mode, the PWM frequency, *fdsm*, is determined by MAIN_DSM.freq.

MAIN_DSM.freq (Byte 0x8045 Bits 0 – 2)	
Value	**Description**
0	*fdsm* = *fcore* / 4
1	*fdsm* = *fcore* / 8
2	*fdsm* = *fcore* / 16
3	*fdsm* = *fcore* / 32
4	*fdsm* = *fcore* / 64
5	*fdsm* = *fcore* / 128
6	*fdsm* = *fcore* / 256
7	Reserved (do not use).

In PWM output mode, MAIN_DSM.clamp determines whether the PWM generator is clamped or unclamped.

MAIN_DSM.clamp (Byte 0x8045 Bit 3)	
Value	**Description**
0	Clamp disabled; PWM wraps around
1	Clamp enabled; PWM output is clamped

MAIN_INTER

MAIN_INTER determines the interpolation factor or output resolution of the iC-TW8. MAIN_INTER is a 16-bit value equal to the number of output increments (pre-AB divider output edges) per Sin/Cos input cycle.

MAIN_INTER (Word 0x8046)	
Value	**Description**
0	MAIN_INTER = 65536 (x16384).
1 – 3	Reserved (do not use)
4 – 65535	Interpolation factors x1 – x16383.75

In AB output mode, the interpolation factor in pre-divider AB output cycles per input cycle, *inter*, is calculated as

$$inter = \frac{\text{MAIN\_INTER}}{4}$$

Thus if MAIN_INTER = 12000, *inter* = x3000.

In PWM output mode (MAIN_OUT.mode = 2), the interpolation factor, *inter*, must be less than or equal to 1024.

MAIN_HYST

MAIN_HYST determines the output hysteresis of the signal path.

MAIN_HYST (Word 0x8048 Bits 0 – 8)	
Value	**Description**
0 – 511	Minimum – maximum input hysteresis

The hysteresis in sensor input (Sin/Cos) degrees, *hyst*, is calculated as

$$hyst = \text{MAIN\_HYST} \cdot \frac{360}{8192}$$

The equivalent hysteresis in output AB edges is a function of the interpolation factor and the AB divider.

MAIN_FLTR

The MAIN_FLTR register contains variables that determine the response of the output filter, which is implemented as a servo loop.

MAIN_FLTR.fb selects whether or not delay is used in the feedback path of the servo loop.

MAIN_FLTR.fb (Byte 0x804A Bit 0)	
Value	**Description**
0	No delay in the feedback path
1	Delay in the feedback path

Delay in the filter feedback path is used to compensate for the lag at constant speed caused by the latency of the iC-TW8 (lag recovery). The amount of delay, *delay*, is calculated as

$$delay = \frac{640}{fcore}$$

For example, if *fcore* = 32 MHz, *delay* = 20µs.

MAIN_FLTR.p determines the proportional gain in the forward path of the filter loop.

MAIN_FLTR.p (Byte 0x804A Bits 1 – 3)	
Value	**Description**
0	*pgain* = 1
1	*pgain* = 0.5
2	*pgain* = 0.25
3	*pgain* = 0.125
4	*pgain* = 0.0625
5	*pgain* = 0.03125
6	*pgain* = 0.015625
7	*pgain* = 0 (do not use)

For 0 ≤ MAIN_FLTR.p ≤ 6, the proportional gain *pgain*, is calculated as

$$pgain = \frac{1}{2^{\text{MAIN\_FLTR.p}}}$$

MAIN_FLTR.i determines the integral gain in the forward path of the filter loop.

MAIN_FLTR.i (Byte 0x804A Bits 4 – 6)	
Value	**Description**
0	*igain* = 1
1	*igain* = 0.5
2	*igain* = 0.25
3	*igain* = 0.125
4	*igain* = 0.0625
5	*igain* = 0.03125
6	*igain* = 0.015625
7	*igain* = 0

For 0 ≤ MAIN_FLTR.i ≤ 6, the integral gain *igain*, is calculated as

$$igain = \frac{1}{2^{\mathrm{MAIN\_FLTR.i}}}$$

The MAIN_FLTR register value necessary to realize a given filter configuration is called its instance. The following instances are recommended.

Recommended Filter Instances				
Instance	**Mode**	**.p**	**.i**	**.fb**
52	PI	2	3	0
70	PI	3	4	0
71	Lag Recovery	3	4	1
88	PI	4	5	0
89	Lag Recovery	4	5	1
106	PI	5	6	0
107	Lag Recovery	5	6	1
112	P	0	7	0
114	P	1	7	0
116	P	2	7	0
118	P	3	7	0
120	P	4	7	0
122	P	5	7	0
124	P	6	7	0

Filter instances other than those shown in the table above are possible, but may result in unstable filter response.

MAIN_OUT

The MAIN_OUT register contains variables that determine the operating output mode and startup delay of the iC-TW8.

MAIN_OUT.start determines the amount of time the iC-TW8 waits during startup for its analog circuitry to settle.

MAIN_OUT.start (Byte 0x804B Bits 0 – 3)	
Value	**Description**
0 – 15	Minimum – Maximum Startup Delay

The actual startup delay *twait* is calculated as

$$twait = \frac{128}{fcore} \cdot \left(2^{\mathrm{MAIN\_OUT.start}+3} - 1\right)$$

For example, if *fcore* = 32 MHz, a MAIN_OUT.start value of 7 produces a startup delay of *twait* = 4.096 milliseconds.

MAIN_OUT.mode determines the output mode of the iC-TW8.

MAIN_OUT.mode (Byte 0x804B Bits 4 – 5)	
Value	**Description**
0	AB output mode
1	Serial-only output mode
2	PWM output mode
3	Reserved (do not use)

In AB output mode, the iC-TW8 outputs industry-standard AB quadrature encoder signals. This is the most common operating mode for using the iC-TW8 in an encoder.

In serial-only output mode, the A, B, and Z outputs are disabled and sensor position (angle) must be read via the SPI or 1-wire serial interfaces.

In PWM output mode, the iC-TW8 outputs a differential pulse-width modulated signal proportional to the sensor position (angle). In this mode, the interpolation factor, *inter*, must be less than or equal to 1024.

MAIN_CLOCK

The MAIN_CLOCK register contains variables that determine the core clock frequency, *fcore*, of the iC-TW8.

The iC-TW8 has an internal oscillator but can also be used with an external crystal or other clock source.

MAIN_CLOCK.xtal (Byte 0x804C Bit 0)	
Value	**Description**
0	Use internal oscillator
1	Use external crystal or oscillator

If MAIN_CLOCK.xtal = 1 and no external clock signal is present the iC-TW8 reverts to using its internal oscillator and sets a fault bit (see FLT_STAT on page 27). Once an external clock signal becomes available, the iC-TW8 switches back to using the external oscillator and clears the fault. When using an external oscillator, *fcore* is the frequency of the external oscillator.

If the internal oscillator is used, MAIN_CLOCK.div can be used to divide its frequency to achieve a lower f_{CORE}.

MAIN_CLOCK.div (Byte 0x804C Bits 1 – 2)	
Value	**Description**
0	*fcore* = internal oscillator/4
1	Reserved (do not use)
2	*fcore* = internal oscillator/2
3	*fcore* = internal oscillator

MAIN_CLOCK.freq is used to tune the frequency of the internal oscillator.

MAIN_CLOCK.freq (Byte 0x804C Bits 3 – 6)	
Value	**Description**
0 – 15	Minimum – Maximum oscillator frequency

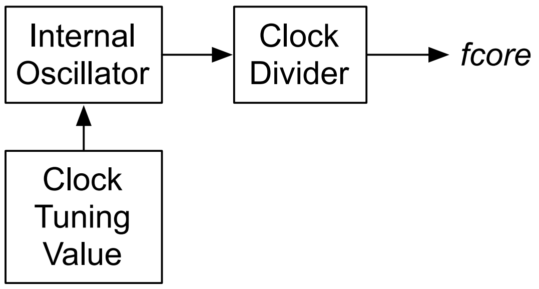

Figure 11: Internal Oscillator Configuration

MAIN_CLOCK.xforce is used to force the iC-TW8 to use the external crystal (if selected) regardless of VDD.

MAIN_CLOCK.xforce (Byte 0x804C Bit 7)	
Value	**Description**
0	Switch to internal oscillator at low VDD
1	Always use external crystal or oscillator

MAIN_CLOCK.xforce should always be set to 1.

MAIN_ZPOS

MAIN_ZPOS determines the position of the Z output pulse within a sensor (Sin/Cos) input cycle when the index sensor connected to the ZERO inputs is active. MAIN_ZPOS is a 14-bit value stored in a word variable.

MAIN_ZPOS (Word 0x804E)	
Value	**Description**
0 – 16383	Z output pulse at 0° – 359.98° input

The actual position of the Z output pulse within a sensor (Sin/Cos) input cycle, *zpos*, is calculated as

$$zpos = \text{MAIN\_ZPOS} \cdot \frac{360}{16384}$$

Thus if MAIN_ZPOS = 8192, *zpos* = 180°.

MAIN_Z

The MAIN_Z register contains variables that determine the ZERO input comparator's switching threshold and whether or not the iC-TW8's internal 32-bit position counter is reset by the Z output or not at all.

MAIN_Z.th determines the switching threshold of the ZERO input comparator. MAIN_Z.th is a signed 4-bit number stored as a 2's complement value.

MAIN_Z.th (Byte 0x8050 Bits 0 – 4)	
Value	**Description**
±15	ZERO input comparator switching threshold

The actual switching threshold of the ZERO input comparator, *zth*, is calculated as

$$zth = \text{MAIN\_Z.th} \cdot 30\text{mV}$$

For example, if MAIN_Z.th = 9, *zth* = +0.27V.

MAIN_Z.reset determines if and when the iC-TW8's internal 32-bit position counter is reset.

MAIN_Z.reset (Byte 0x8050 Bit 5)	
Value	**Description**
0	Reset position counter at *zpos*
1	Do not reset position counter on Z

If MAIN_Z.reset = 0, the internal 32-bit position counter is reset at sensor input position (angle) *zpos* whenever the ZERO input is active.

If MAIN_Z.reset = 1, the internal 32-bit position counter is not reset during operation.

AB_CFG0

The AB_CFG0 register contains variables that determine the startup mode and polarity of the AB and Z outputs.

AB_CFG0.dir determines the counting direction of the AB outputs.

AB_CFG0.dir (Byte 0x8052 Bit 0)	
Value	**Description**
0	Normal counting direction
1	Reversed counting direction

AB_CFG0.apol determines whether the polarity of the A output is normal or inverted.

AB_CFG0.apol (Byte 0x8052 Bit 1)	
Value	**Description**
0	Normal A polarity and A output low at *zpos*.
1	Inverted A polarity and A output high at *zpos*.

If AB_CFG0.startmode = 0, AB_CFG0.apol determines the state of the A output after startup. If AB_CFG0.startmode = 1 or 3, AB_CFG1 = 0 (post-AB divider disabled), and *inter* (the interpolation factor) is an integer, AB_CFG0.apol also determines the polarity (state) of the A output at sensor position (angle) *zpos*.

Note that setting apol also reverses the counting direction of the AB outputs. In this case, use AB_CFG0.dir to restore the original AB output counting direction. Setting both apol and bpol does not change the AB output counting direction.

AB_CFG0.bpol determines whether the polarity of the B output is normal or inverted.

AB_CFG0.bpol (Byte 0x8052 Bit 2)	
Value	**Description**
0	Normal B polarity and B output low at *zpos*.
1	Inverted B polarity and B output high at *zpos*.

If AB_CFG0.startmode = 0, AB_CFG0.bpol determines the state of the B output after startup. If AB_CFG0.startmode = 1 or 3, AB_CFG1 = 0 (post-AB divider disabled), and *inter* (the interpolation factor) is an integer, AB_CFG0.bpol also determines the polarity (state) of the B output at sensor position (angle) *zpos*.

Note that setting bpol also reverses the counting direction of the AB outputs. In this case, use AB_CFG0.dir to restore the original AB output counting direction. Setting both apol and bpol does not change the AB output counting direction.

AB_CFG0.zpol determines the polarity (active state) of the Z output.

AB_CFG0.zpol (Byte 0x8052 Bit 3)	
Value	**Description**
0	Z output is active high.
1	Z output is active low.

AB_CFG0.startmode determines how the AB and Z outputs behave at startup.

AB_CFG0.startmode (Byte 0x8052 Bits 5 – 6)	
Value	**Description**
0	Relative startup mode
1	Same Phase startup mode
2	Reserved (do not use)
3	Absolute Burst startup mode

In Relative startup mode, on startup the A and B outputs are in the states determined by AB_CFG0 .apol and bpol (respectively) regardless of the sensor position (angle).

In Same Phase startup mode, if AB_CFG1 = 0 (post-AB divider disabled), and *inter* (the interpolation factor) is an integer, then the phase relationship of the AB and Z outputs is the same after every startup as determined by AB_CFG0.apol, bpol, and zpol. This mode is the most similar to the operation of a non-interpolated encoder. In Same Phase startup mode, if AB_CFG1 $\neq$ 0 (post-AB divider enabled), or *inter* (the interpolation factor) is not an integer, then the phase relationship of the AB and Z outputs may not be the same after every startup.

Absolute Burst startup mode is like Same Phase, except that the sensor position (angle) within one input cycle is counted out on the A and B outputs at startup. A maximum of $2 \cdot inter$ AB edges (±180°) may be generated at a rate determined by AB_VTOP at startup.

AB_CFG1

The AB_CFG1 register contains the post-AB divider variable. The post-AB divider allows the iC-TW8 to produce fractional interpolation factors.

AB_CFG1.div (Byte 0x8053 Bits 0 – 4)	
Value	**Description**
0	Post-AB Divider disabled
1 – 31	Minimum – maximum Post-AB Divider value

The actual value used by the post-AB divider, *div*, is calculated as

$$div = \text{AB\_CFG1.div+1}$$

When using the post-AB divider, the effective interpolation factor, *intereff*, is

$$intereff = \frac{inter}{div}$$

For example, if *inter* = 4096 and *div* = 20, the effective interpolation factor is 204.8.

AB_ZWIDTH

The AB_ZWIDTH register determines the width of the Z output pulse.

AB_ZWIDTH (Byte 0x8054)	
Value	**Description**
0 – 255	Minimum – maximum Z output pulse width

AB_ZWIDTH must be less than MAIN_INTER.

The actual width of the Z output pulse, *zwidth*, is $zwidth = \text{AB\_ZWIDTH+1}$ pre-divider AB output edges. If the post-AB divider is used,

$$zwidth = \frac{\text{AB\_ZWIDTH+1}}{div}$$

AB_VTOP

The AB_VTOP register is used to limit the maximum AB output frequency.

AB_VTOP (Byte 0x8055 Bits 0 – 6)	
Value	**Description**
0 – 127	Maximum – minimum AB output frequency

The actual AB output frequency limit *fab*, is calculated as

$$fab = \frac{fcore}{4 \cdot (\text{AB\_VTOP}+1) \cdot div}$$

For example, if *fcore* = 32 MHz, the post-AB divider is not used (*div* = 1), and AB_VTOP = 39, *fab* is limited to 200 kHz. If the sensor input speed increases beyond this point, the AB output frequency is clamped to *fab* and a fault status bit is set. See FLT_STAT on page 27 for more information.

Because edge separation (the time between two consecutive AB edges) is the reciprocal of the AB output frequency, AB_VTOP can also be used to set the minimum edge separation of the AB outputs. The minimum edge separation time *tedgemin*, is calculated as

$$tedgemin = \frac{(\text{AB\_VTOP}+1) \cdot div}{fcore}$$

For example, if *fcore* = 32 MHz, the post-AB divider is not used (*div* = 1), and AB_VTOP = 39, *tedgemin* is 1.25 µs. If the sensor input speed increases beyond this point, the AB output edges remain spaced at *tedgemin* and a fault status bit is set. See FLT_STAT on page 27 for more information.

If there is noise on the input signals, it is possible that the AB outputs can dither at standstill, especially if the hysteresis is not set correctly. The time between dithering edges *tdither*, is calculated as

$$tdither = \frac{(\text{AB\_VTOP}+1)}{fcore} = \frac{tedgemin}{div}$$

If $div = 1$, $tdither = tedgemin$. However, if $\text{div} \neq 1$, $tdither < tedgemin$,

ADPT_CFG

The ADPT_CFG register contains bits that determine which signal path parameters are adapted during operation, whether the LUT is used, and whether adapted parameter values are stored to EEPROM during operation.

ADPT_CFG.doff enables auto adaption of digital offset.

ADPT_CFG.doff (Byte 0x8058 Bit 0)	
Value	**Description**
0	Disable digital offset adaption
1	Enable digital offset adaption

ADPT_CFG.dgain enables auto adaption of digital gain match.

ADPT_CFG.dgain (Byte 0x8058 Bit 1)	
Value	**Description**
0	Disable digital gain match adaption
1	Enable digital gain match adaption

ADPT_CFG.dphase enables auto adaption of digital phase.

ADPT_CFG.dphase (Byte 0x8058 Bit 2)	
Value	**Description**
0	Disable digital phase adaption
1	Enable digital phase adaption

ADPT_CFG.aoff enables auto adaption of analog offset.

ADPT_CFG.aoff (Byte 0x8058 Bit 3)	
Value	**Description**
0	Disable analog offset adaption
1	Enable analog offset adaption

If analog offset adaption is enabled, digital offset adaption must also be enabled.

ADPT_CFG.again enables auto adaption of analog gain.

ADPT_CFG.again (Byte 0x8058 Bit 4)	
Value	**Description**
0	Disable analog gain adaption
1	Enable analog gain adaption

If analog gain adaption is enabled, digital gain match adaption and digital offset adaption must also be enabled.

ADPT_CFG.lut determines whether or not the sensor distortion correction look-up table (LUT) is used.

ADPT_CFG.lut (Byte 0x8058 Bit 5)	
Value	**Description**
0	Disable LUT
1	Enable LUT

ADPT_CFG.store determines whether or not adapted parameter values are automatically stored to the EEPROM during operation (auto store).

ADPT_CFG.store (Byte 0x8058 Bit 6)	
Value	**Description**
0	Disable auto store
1	Enable auto store

ADPT_DETAIL

The ADPT_DETAIL register contains variables and bits that determine the auto-adaption speed limit, the auto-adaption correction rate, and whether or not auto-adaption is disabled in the event of an ADC fault.

ADPT_DETAIL.flimit determines the highest sensor input frequency at which auto-adaption occurs.

ADPT_DETAIL.flimit (Byte 0x8059 Bits 0 – 2)	
Value	**Description**
0 – 7	Minimum – maximum auto-adaption sensor input frequency limit

The actual sensor input auto-adaption frequency limit, *fadapt*, is calculated as

$$fadapt = \frac{fcore}{83886.08} \cdot 2^{\text{ADPT\_DETAIL.flimit}}$$

During operation, auto-adaption occurs at sensor input frequencies, *finput*, of

$$0 < finput \le fadapt$$

For example, if ADPT_DETAIL.flimit = 5 and *fcore* = 32 MHz, *fadapt* = 12.2 kHz and auto-adaption ceases at sensor input frequencies above 12.2 kHz.

ADPT_DETAIL.tbase determines the time-base used to apply auto-adaption corrections.

ADPT_DETAIL.tbase (Byte 0x8059 Bits 3 – 5)	
Value	**Description**
0 – 7	Minimum – maximum correction time-base

The actual time-base used to apply auto-adaption corrections, *tbase*, is calculated as

$$tbase = \frac{2^{\text{ADPT\_DETAIL.tbase}+7}}{fcore}$$

For example, if ADPT_DETAIL.tbase = 4 and *fcore* = 32 MHz, *tbase* = 64µs.

ADPT_DETAIL.fault determines whether or not auto-adaption continues if an ADC fault occurs.

ADPT_DETAIL.fault (Byte 0x8059 Bit 6)	
Value	**Description**
0	Continue auto-adaption during ADC fault
1	Disable auto-adaption during ADC fault

ADPT_STORE

The ADPT_STORE register contains variables that determine the thresholds for storing adapted parameter values to EEPROM during operation. If auto store is not used (ADPT_CFG.store = 0), these values have no effect.

ADPT_store.offth determines the threshold for storing adapted digital offset values to EEPROM during operation.

ADPT_STORE.offth (Byte 0x805A Bits 0 – 3)	
Value	**Description**
0 – 15	Minimum – maximum offset threshold

The actual threshold value in ADC increments, *offth*, is calculated as

$$offth = 2^{\frac{\text{ADPT\_STORE.offth}}{2}+1}$$

An auto store EEPROM write occurs if either

$$\left|PAR\_DOFS - PAR\_BAK\_DOFS\right| > offth$$
$$\left|PAR\_DOFC - PAR\_BAK\_DOFC\right| > offth$$

For example, if ADPT_STORE.offth = 9, *offth* = 45.

ADPT_store.gainth determines the threshold for storing adapted digital gain values to EEPROM during operation.

ADPT_STORE.gainth (Byte 0x805A Bits 4 – 7)	
Value	**Description**
0 – 15	Minimum – maximum gain threshold

The actual threshold value, *gainth*, is calculated as

$$gainth = 2^{\frac{\text{ADPT\_STORE.gainth}}{2}+1}$$

An auto store EEPROM write occurs if

$$\left|PAR\_DGAIN - PAR\_BAK\_DGAIN\right| > gainth$$

For example, if ADPT_STORE.gainth = 7, *gainth* = 22 which is equivalent to a change in gain of 0.005.

ADPT_CORR

The ADPT_CORR register contains variables that control auto-adaption correction.

ADPT_CORR.prop determines the response of the auto-adaption correction loop

ADPT_CORR.prop (Byte 0x805B Bits 0 – 1)	
Value	**Description**
0	Linear correction (1 increment/*tbase*)
1	Slow exponential correction (*prop* = 0.25)
2	Medium exponential correction (*prop* = 0.5)
3	Fast exponential correction (*prop* = 0.75)

Smaller ADPT_CORR.prop values apply the auto-adaption corrections more smoothly, resulting in less disturbance to the AB outputs.

ADPT_CORR.offtol determines the maximum uncorrected (residual) digital offset that is tolerated before correction is applied.

ADPT_CORR.offtol (Byte 0x805B Bits 2 – 4)	
Value	**Description**
0 – 7	Minimum – maximum digital offset tolerance

The actual tolerance value in ADC increments, *offtol*, is calculated as

$$offtol = 2^{\frac{\text{ADPT\_CORR.offtol}}{2}+1}$$

If the residual (uncorrected) digital offset of both sensor channels is ≤ *offtol*, no correction is applied. If the residual (uncorrected) digital offset of either sensor channel is > *offtol*, correction is applied to that channel. Thus, during operation the residual (uncorrected) digital offset of both sensor channels is always ≤ *offtol*.

For example, if ADPT_STORE.offtol = 2, *offtol* = 4. In this case, changes in digital offsets over 4 ADC increments will be corrected and the residual (uncorrected) digital offset of both sensor channels during operation will always be ≤ 4 (977µV at the ADC inputs).

ADPT_CORR.gaintol determines the maximum uncorrected (residual) digital gain mismatch that is tolerated before correction is applied.

ADPT_CORR.gaintol (Byte 0x805B Bits 5 – 7)	
Value	**Description**
0 – 7	Minimum – maximum digital gain tolerance

The actual tolerance value, *gaintol*, is calculated as

$$gaintol = 2^{\frac{\text{ADPT\_CORR.gaintol}}{2}+1}$$

If the residual (uncorrected) digital gain mismatch is ≤ *gaintol*, no correction is applied; if the residual (uncorrected) digital gain mismatch is > *gaintol*, correction is applied. Thus, during operation the residual (uncorrected) digital gain mismatch is always ≤ *gaintol*.

For example, if ADPT_CORR.gaintol = 7, *gainth* = 22. In this case, changes in digital gain mismatch over 22 ADC increments will be corrected and the residual (uncorrected) digital gain mismatch during operation will always be ≤ 22 (0.005).

MON_CFG

The MON_CFG register contains bits that control the ADC and adaption quality monitors and the configuration of the STATUS output (pin 19). See VAR Block Registers on page 33 for more information on using the quality monitors.

MON_CFG.adc determines whether or not the ADC quality monitor is used.

MON_CFG.adc (Byte 0x805D Bit 0)	
Value	**Description**
0	Disable ADC quality monitor
1	Enable ADC quality monitor

If MON_CFG.adc = 0 the ADC quality monitor is disabled and the ADC underflow fault level (see page 33) is set to

$$\left|\begin{matrix}\text{Sin or Cos}\\ \text{Input Voltage}\end{matrix}\right| \le \frac{148\text{mV}}{again}$$

If MON_CFG.adc = 1 the ADC quality monitor is enabled and the ADC underflow fault level (see page 33) is set to

$$\left|\begin{matrix}\text{Sin or Cos}\\ \text{Input Voltage}\end{matrix}\right| \le \frac{adcflt}{128 \cdot again}$$

adcflt is determined by variables in the MON_ADC register (see page 24).

MON_CFG.adapt determines whether or not the adaption quality monitor is used.

MON_CFG.adapt (Byte 0x805D Bit 1)	
Value	**Description**
0	Disable adaption quality monitor
1	Enable adaption quality monitor

If MON_CFG.adapt = 0, the adaption quality monitor is disabled, but maximum adaption is still limited by *offflt*, *gainflt*, and *phflt*. If MON_CFG.adapt = 1, the adaption quality monitor is enabled and adaption fault levels and maximum adaption are set by *offflt*, *gainflt*, and *phflt*. *offflt* is determined by variables in the MON_OFF register (page 25), *gainflt* is determined by variables in the MON_GAIN register (page 25), and *phflt* is determined by variables in the MON_PHASE register (page 26).

MON_CFG.pwm determines the characteristics of the PWM signal used to drive the STATUS output.

MON_CFG.pwm (Byte 0x805D Bits 2 – 3)	
Value	**Description**
0	STATUS dutycycle is *pwm*/255
1	STATUS dutycycle is 1 – *pwm*/255
2	STATUS dutycycle is $2^{\frac{pwm}{32}} / 255$
3	STATUS dutycycle is $1 - 2^{\frac{pwm}{32}} / 255$

MON_CFG.pwm = 0 or 1 provide a linear PWM output on the status pin indicating ADC and/or adaption quality. MON_CFG.pwm = 2 or 3 provide a logarithmic PWM output which is useful for driving an LED. The PWM signal, *pwm*, is calculated from the quality monitors as

$$pwm = \text{QM\_ADC} \cdot \text{MON\_CFG.adc} + \text{QM\_ADAPT} \cdot \text{MON\_CFG.adapt}$$

MON_CFG.adcth is the most significant bit of MON_ADC.th.

MON_CFG.adcth (Byte 0x805D Bit 6)	
Value	**Description**
0	$0 \le adcth < 120$
1	$120 \le adcth < 184$

MON_ADC

The MON_ADC register contains variables that determine the ADC (Analog to Digital Converter) quality threshold and ADC underflow fault level. If ADC quality monitoring is disabled (MON_CFG.adc = 0), these values are ignored. See VAR Block Registers on page 33 for more information on using the quality monitors.

MON_ADC.limit and MON_ADC.th together determine the ADC level that causes an ADC underflow fault.

MON_ADC.limit (Byte 0x805E Bits 0 – 3)	
Value	**Description**
0 – 15	Highest – lowest ADC underflow limit

The actual ADC underflow limit, *adclim*, is calculated as

$$adclim = 2^{\frac{\text{MON\_ADC.limit}}{2}+1}$$

MON_ADC.th determines the threshold between acceptable and marginal ADC level for the ADC quality monitor.

MON_ADC.th (Byte 0x805E Bits 4 – 7)	
Value	**Description**
0 – 15	Lowest – highest ADC quality threshold

The actual ADC quality threshold level, *adcth*, is calculated as

$$adcth = 4 \cdot \left(\begin{array}{l} \text{MON\_ADC.th} + \\ 16 \cdot \text{MON\_CFG.adcth} \end{array} \right) + 60$$

Note that MON_ADC.th is actually a 5-bit variable with its most significant bit stored in MON_CFG.adcth.

The actual ADC underflow fault level, *adcflt*, is

$$adcflt = adcth - adclim$$

MON_OFF

The MON_OFF register contains variables that determine the maximum adaption and quality threshold levels for digital offset adaption.

MON_OFF.limit and MON_OFF.th together determine the maximum amount of digital offset adaption. Reaching maximum adaption can be configured to cause an adaption fault (see page 28).

MON_OFF.limit (Byte 0x805F Bits 0 – 3)	
Value	**Description**
0 – 12	Lowest – highest offset adaption limit
13 – 15	Reserved (do not use)

The actual digital offset adaption limit, *offlim*, is calculated as

$$offlim = 2^{\text{MON\_OFF.limit}}$$

MON_OFF.th determines the threshold between acceptable and excessive adaption.

MON_OFF.th (Byte 0x805F Bits 4 – 7)	
Value	**Description**
0 – 12	Lowest – highest offset adaption quality threshold
13 – 15	Reserved (do not use)

The actual digital offset adaption quality threshold level, *offqth*, is calculated as

$$offqth = 2^{\text{MON\_OFF.th}}$$

The actual maximum amount of digital offset adaption and the offset adaption fault level, *offflt*, is calculated as

$$offflt = offqth + offlim$$

Digital offset adaption ceases whenever

$$\left|\text{DOFFS} - \text{PAR\_BASE\_DOFFS}\right| \geq offflt$$
$$\left|\text{DOFFC} - \text{PAR\_BASE\_DOFFC}\right| \geq offflt$$

MON_GAIN

The MON_GAIN register contains variables that determine the maximum adaption and quality threshold levels for digital gain match adaption.

MON_GAIN.limit and MON_GAIN.th together determine the maximum amount of digital gain match adaption. Reaching maximum adaption can be configured to cause an adaption fault (see page 28).

MON_GAIN.limit (Byte 0x8060 Bits 0 – 3)	
Value	**Description**
0 – 12	Lowest – highest gain match adaption limit
13 – 15	Reserved (do not use)

The actual digital gain match adaption limit, *gainlim*, is calculated as

$$gainlim = 2^{\text{MON\_GAIN.limit}}$$

MON_GAIN.th determines the threshold between acceptable and excessive adaption.

MON_GAIN.th (Byte 0x8060 Bits 4 – 7)	
Value	**Description**
0 – 12	Lowest – highest gain match adaption quality threshold
13 – 15	Reserved (do not use)

The actual digital gain match adaption quality threshold level, *gainqth*, is calculated as

$$gainqth = 2^{\text{MON\_GAIN.th}}$$

The actual maximum amount of digital gain match adaption and the gain match adaption fault level, *gainflt*, is calculated as

$$gainflt = gainqth + gainlim$$

Digital gain match adaption ceases whenever

$$\left|\text{DGAIN} - \text{PAR\_BASE\_DGAIN}\right| \geq gainflt$$

MON_PHASE

The MON_PHASE register contains variables that determine the maximum adaption, quality threshold, and fault level for digital phase adaption.

MON_PHASE.limit and MON_PHASE.th together determine the maximum amount of digital phase adaption. Reaching maximum adaption can be configured to cause an adaption fault (see page 28).

MON_PHASE.limit (Byte 0x8061 Bits 0 – 3)	
Value	**Description**
0 – 12	Lowest – highest phase adaption limit
13 – 15	Reserved (do not use)

The actual digital phase adaption limit, *phlim*, is calculated as

$$phlim = 2^{\text{MON\_PHASE.limit}}$$

MON_PHASE.th determines the threshold between acceptable and excessive adaption.

MON_PHASE.th (Byte 0x8061 Bits 4 – 7)	
Value	**Description**
0 – 12	Lowest – highest phase adaption quality threshold
13 – 15	Reserved (do not use)

The actual digital phase adaption quality threshold level, *phqth*, is calculated as

$$phqth = 2^{\text{MON\_PHASE.th}}$$

The actual maximum amount of digital phase adaption and the phase adaption fault level, *phflt*, is calculated as

$$phflt = phqth + phlim$$

Digital phase adaption ceases whenever

$$\left|\text{DPH} - \text{PAR\_BASE\_DPH}\right| \geq phaseflt$$

FLT_CFG

The FLT_CFG register contains bits that control the behavior of the FAULT output (pin 20).

FLT_CFG.vwarn determines whether or not operational warnings are treated as faults.

FLT_CFG.vwarn (Byte 0x8062 Bit 0)	
Value	**Description**
0	Operational warnings ignored
1	Operational warnings activate FAULT output

FLT_CFG.hold determines whether or not the AB outputs are disabled when a fault occurs.

FLT_CFG.hold (Byte 0x8062 Bits 1 – 2)	
Value	**Description**
0	AB outputs operational during a fault
1	Reserved (do not use)
2	Reserved (do not use)
3	AB outputs disabled during a fault

FLT_CFG.pol determines the polarity of the FAULT output (pin 20).

FLT_CFG.pol (Byte 0x8062 Bit 3)	
Value	**Description**
0	FAULT output is active low
1	FAULT output is active high

By default, the FAULT output is active low at startup. After a successful EEPROM read, FLT_CGF.pol is programmed to the polarity determined by FLT_CFG.pol. Thus, if FLT_CFG.pol = 1, faults which occur before a successful EEPROM read after startup are not indicated at the programmed polarity.

FLT_CFG.long determines whether or not the FAULT output is prolonged to allow easier detection of transient faults.

FLT_CFG.long (Byte 0x8062 Bit 4)	
Value	**Description**
0	FAULT output active only while fault active
1	FAULT output activation prolonged

If FLT_CFG.long = 1, the actual time by which the FAULT output is prolonged is calculated as

$$long[ms] = \frac{32}{fcore[\mathrm{MHz}]} \cdot 20$$

For example, if FLT_CFG = 1 and *fcore* = 32 MHz, the FAULT output remains active for 20 ms longer that the fault condition persists.

FLT_STAT

The FLT_STAT register contains bits that are dynamically updated during operation to indicate fault status. FLT_STAT register bits are "sticky" so that once activated, they stay activated even if the specific condition is subsequently resolved. Active FLT_STAT bits may be cleared by writing 0x00 to the FLT_STAT register.

FLT_STAT.ee indicates whether or not an EEPROM fault has occurred.

FLT_STAT.ee (Byte 0x8064 Bit 0)	
Value	**Description**
0	No EEPROM fault
1	EEPROM fault occurred

If FLT_STAT.ee = 1, the specific EEPROM fault can be determined by reading the STAT_EE register in the VAR block (page 35). Note that a checksum error must be indicated for *both* PAR block 0 (STAT_EE.par0 = 1) and PAR block 1 (STAT_EE .par1 = 1) to activate FLT_STAT.ee.

FLT_STAT.adc indicates whether or not an ADC fault has occurred.

FLT_STAT.adc (Byte 0x8064 Bit 1)	
Value	**Description**
0	No ADC fault
1	ADC fault occurred

If FLT_STAT.adc = 1, the specific ADC fault can be determined by reading the STAT_SP register in the VAR block (page 33).

FLT_STAT.adapt indicates whether or not an adaption fault has occurred.

FLT_STAT.adapt (Byte 0x8064 Bit 2)	
Value	**Description**
0	No adaption fault
1	Adaption fault occurred

FLT_STAT.adapt is set by STAT_SP.adapt; see STAT_SP on page 33 for more information.

FLT_STAT.vwarn indicates whether or not an overspeed warning has occurred.

FLT_STAT.vwarn (Byte 0x8064 Bit 3)	
Value	**Description**
0	No overspeed warning
1	Overspeed warning

If FLT_STAT.vwarn = 1, instantaneous AB output frequency is above fab or $\frac{fcore}{512}$. See AB_VTOP on page 20 for more information on fab.

FLT_STAT.vfatal indicates whether or not a fatal overspeed fault has occurred. A fatal overspeed fault may cause undefined AB outputs and erroneous position.

FLT_STAT.vfatal (Byte 0x8064 Bit 4)	
Value	**Description**
0	No fatal overspeed fault
1	Fatal ovverspeed fault

If FLT_STAT.vfatal = 1, instantaneous AB output frequency is above $\frac{fcore}{256}$, the filter lag is too large, or the AB output is more than half an input cycle behind the sensor input position due to prolonged operation above fab (see page 20).

FLT_STAT.intern indicates whether or not an internal fault in the iC-TW8 has occurred.

FLT_STAT.intern (Byte 0x8064 Bit 6)	
Value	**Description**
0	No internal iC-TW8 fault
1	Internal iC-TW8 fault

If FLT_STAT.intern = 1, the external crystal is selected but not working or there was a timeout fault on the serial communication interface. The specific fault can be determined by reading the STAT_SP register in the VAR block (page 33).

FLT_EN

The FLT_EN register contains bits that determine which faults activate or latch the FAULT output (pin 20).

FLT_EN.ee determines whether or not an EEPROM fault latches the FAULT output.

FLT_EN.ee (Byte 0x8066 Bit 0)	
Value	**Description**
0	EEPROM fault does not affect FAULT output
1	EEPROM fault latches FAULT output.

FLT_EN.adc determines whether an ADC fault latches the FAULT output.

FLT_EN.adc (Byte 0x8066 Bit 1)	
Value	**Description**
0	ADC fault activates FAULT output
1	ADC fault latches FAULT output

FLT_EN.adapt determines whether or not an adaption fault latches the FAULT output.

FLT_EN.adapt (Byte 0x8066 Bit 2)	
Value	**Description**
0	Adaption fault does not affect FAULT output
1	Adaption fault latches FAULT output

FLT_EN.vwarn determines whether or not an overspeed warning latches the FAULT output.

FLT_EN.vwarn (Byte 0x8066 Bit 3)	
Value	**Description**
0	Overspeed warning activates FAULT output (only if FLT_CFG.vwarn = 1)
1	Overspeed warning latches FAULT output

If FLT_EN.vwarn = 1, overspeed warnings are treated as faults.

FLT_STAT.vfatal determines whether or not a fatal overspeed fault latches the FAULT output. A fatal overspeed fault may cause undefined AB outputs and erroneous position.

FLT_EN.vfatal (Byte 0x8066 Bit 4)	
Value	**Description**
0	Fatal overspeed fault does not affect FAULT output
1	Fatal overspeed fault latches FAULT output

FLT_EN.intern determines whether or not an internal fault in the iC-TW8 latches the FAULT output.

FLT_EN.intern (Byte 0x8066 Bit 6)	
Value	**Description**
0	Internal iC-TW8 fault activates FAULT output
1	Internal iC-TW8 fault latches FAULT output

PAR Block Registers

The PAR block contains the signal path parameters for gain, offset correction, and phase correction. The signal path parameters are automatically set during device calibration and can be automatically adjusted during operation to maintain optimum performance.

The PAR block keeps three copies of all parameter values in three identically-structured sub-blocks as explained on page 8. It is only necessary to directly read or write the PAR block registers in the PAR_NOW sub-block. Access to the other sub-blocks is handled automatically by the iC-TW8.

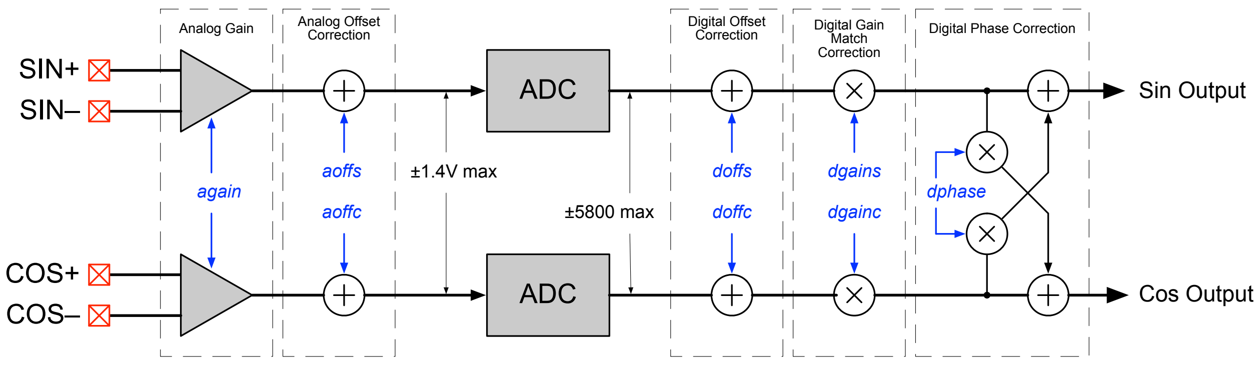

Figure 12: Signal Path

AGAIN (Analog Gain)

The AGAIN register determines the gain used by the differential input amplifiers for the sensor Sin and Cos inputs. The same gain is always applied to both channels.

AGAIN (Byte 0x8074)	
Value	**Description**
0 – 1	Reserved (do not use)
2 – 15	Minimum – maximum analog gain
16 – 255	Reserved (do not use)

The actual analog gain, *again*, is calculated as

$$again[dB] = 3 \cdot \text{AGAIN}$$
$$again = 10^{\frac{3 \cdot \text{AGAIN}}{20}}$$

giving an input gain range of 6 – 45dB (2 – 178) in 3dB steps.

AOFFS, AOFFC (Analog Offset)

The AOFFS and AOFFC registers determine the gross offset correction applied to the output of the analog sin and cos amplifiers respectively. AOFFS and AOFFC are signed values in 2's complement format.

AOFFS (Byte 0x8075) AOFFC (Byte 0x8076)	
Value	**Description**
±31	Analog offset correction
±32 – ±127	Reserved (do not use)

The actual analog offset corrections applied, *aoffs* and *aoffc*, are calculated as

$$aoffs[mV] = \text{AOFFS} \cdot 100$$
$$aoffc[mV] = \text{AOFFC} \cdot 100$$

The effective analog offset correction referenced to the input Sin/Cos signals, *aoffins* and *aoffinc* can be calculated as

$$aoffins[mV] = \frac{\text{AOFFS} \cdot 100}{10^{\frac{3 \cdot \text{AGAIN}}{20}}}$$
$$aoffinc[mV] = \frac{\text{AOFFC} \cdot 100}{10^{\frac{3 \cdot \text{AGAIN}}{20}}}$$

DOFFS, DOFFC (Digital Offset)

The DOFFS and DOFFC registers determine the fine offset correction applied to the output of the Sin and Cos ADCs (Analog to Digital Converters) respectively. DOFFS and DOFFC are signed values in 2's complement format.

DOFFS (Word 0x806E)	
DOFFC (Word 0x8070)	
Value	**Description**
±511	Digital offset correction
±512 – ±32767	Reserved (do not use)

The actual digital offset corrections applied, *doffs* and *doffc*, are calculated as

$$doffs[mV] = \frac{\text{DOFFS}}{4.096}$$
$$doffc[mV] = \frac{\text{DOFFC}}{4.096}$$

giving an offset correction range of ±125mV. The effective offset correction referenced to the input Sin/Cos signals can be calculated as

$$doffins[mV] = \frac{\text{DOFFS}}{4.096 \cdot 10^{\frac{3 \cdot \text{AGAIN}}{20}}}$$
$$doffinc[mV] = \frac{\text{DOFFC}}{4.096 \cdot 10^{\frac{3 \cdot \text{AGAIN}}{20}}}$$

DGAIN (Digital Gain Match)

The DGAIN register determines the gain applied to either the digital sin or cos channel to equalize the signal amplitudes. DGAIN does not apply additional overall gain, it is used only to correct a gain mismatch between the two channels. DGAIN is a signed value in 2's complement format.

DGAIN (Word 0x806C)	
Value	**Description**
±1023	Digital gain match correction
±1024 – ±32767	Reserved (do not use)

If DGAIN < 0, then the digital gain match correction is applied to the sin channel and the cos channel has a gain of 1. In this case, the actual digital gains applied, *dgains* and *dgainc*, are calculated as

$$dgains = 1 - \frac{\text{DGAIN}}{4096}$$
$$dgainc = 1$$

If DGAIN ≥ 0, then the digital gain match correction is applied to the cos channel and the sin channel has a gain of 1. In this case, the actual digital gains applied, *dgains* and *dgainc*, are calculated as

$$dgains = 1$$
$$dgainc = 1 + \frac{\text{DGAIN}}{4096}$$

This gives a gain match correction range of ±1.25.

DPH (Digital Phase)

The DPH register determines the phase correction applied to the digital sin and cos channels to achieve 90° phase shift. DPH is a signed value in 2's complement format.

DPH (Word 0x8072)	
Value	**Description**
±1023	Digital phase correction
±1024 – ±32767	Reserved (do not use)

The actual digital phase correction applied, *dphase*, is calculated as

$$dphase[°] = 2 \cdot \arctan\left(\frac{\text{DPH}}{2048}\right)$$

giving a phase correction range of ±53° of an input cycle.

VAR Block Registers

The VAR block contains the Command (CMD) register, the signal path status register, the EEPROM status register, and the iC-TW8's working variable values as explained on page 9. Except for the CMD, STAT_SP, and STAT_EE registers, VAR block registers should only be read, not written to.

CMD (Command Register)

The CMD register is used to tell the iC-TW8 to update the device configuration, write the CFG and/or PAR blocks to the EEPROM, compensate internal amplifier offsets, or restart. To initiate a command, write the appropriate value to the CMD register. When the command has been executed, the iC-TW8 resets the CMD register to 0x00.

Always wait until the CMD register reads 0x00 before writing a new command. The CMD register may be polled at any time to determine the status of command execution. Do not write more than one command to the CMD register at a time.

CMD (Byte 0x80B0)	
Value	**Description**
0x00	Command Register ready (read only)
0x01	Reserved (do not use)
0x02	Update signal path configuration
0x04	Write All
0x08	Write Parameters
0x10	Reserved (do not use)
0x20	Reserved (do not use)
0x40	Restart
0x80	Reserved (do not use)

The Update command (write 0x02 to 0x80B0) updates the iC-TW8 configuration per the current CFG block register values. Any changes to the CFG block registers (see page 13) must be followed by an Update command to take effect.

The Write All command (write 0x04 to 0x80B0) writes the complete CFG, LUT, and PAR blocks to EEPROM as explained on page 11.

The Write Parameters command (write 0x08 to 0x80B0) writes the PAR_NOW sub-block to EEPROM as explained on page 11.

The Restart command (write 0x40 to 0x80B0) restarts the iC-TW8 as if power had been cycled. The 1-Wire port is inoperative while the Restart command is active.

STAT_SP (Signal Path Status Register)

The STAT_SP register contains bits that indicate the status of the signal path. It can be read to determine the cause of a fault or other condition. Some STAT_SP register bits can be configured as "sticky" so that once activated, they stay activated even if the specific condition is subsequently resolved. Sticky STAT_SP bits can be cleared by writing 0x00 to the STAT_SP register, clearing *all* sticky bits. See FLT_EN on page 28 for information on configuring sticky STAT_SP bits.

STAT_SP (Word 0x809A)		
Bit	**Name**	**Description**
0	adcof	ADC overflow condition
1	adcuf	ADC underflow condition
2	adapt	Excessive adaption
3	vtop	Speed limit exceeded
4	ospeed	Overspeed warning
5	fatal	Fatal overspeed fault
6	lag	Excessive filter lag
7	ab	Excessive AB output lag
8		Reserved
9	1wire	1-wire interface time-out
10		Reserved
11	xtal	External crystal fault
12 – 15		Reserved

STAT_SP.adcof indicates that the signal input level is too high. Specifically, if STAT_SP.adcof = 1,

$$\left|\begin{matrix}\text{Sin or Cos}\\ \text{Input Voltage}\end{matrix}\right| \geq \frac{1.5\text{V}}{again}$$

This is not a fatal fault, but interpolation accuracy is reduced in this condition. See page 30 for the definition of *again*.

If FLT_EN.adc = 1, STAT_SP.adcof is "sticky" and stays activated even if the input signal amplitude is subsequently reduced. See FLT_EN on page 28 for information on FLT_EN.adc.

STAT_SP.adcuf indicates that the signal input level is too low. Specifically, if STAT_SP.adcuf = 1,

$$\left|\begin{array}{c}\text{Sin or Cos}\\\text{Input Voltage}\end{array}\right| \le \frac{148\text{mV}}{again}$$

or

$$\left|\begin{array}{c}\text{Sin or Cos}\\\text{Input Voltage}\end{array}\right| \le \frac{adcflt}{128 \cdot again}$$

This is not a fatal fault, but interpolation accuracy is reduced in this condition. See page 30 for the definition of *again* and page 24 for the definition of *adcflt*.

If FLT_EN.adc = 1, STAT_SP.adcuf is "sticky" and stays activated even if the input signal amplitude is subsequently increased. See FLT_EN on page 28 for information on FLT_EN.adc.

If adaption quality monitoring is enabled (MON_CFG.adapt = 1), STAT_SP.adapt indicates that one or more of the signal path parameters has been changed too much by auto-adaption. During operation, the iC-TW8 continually compares the current value of the signal path parameters in the PAR_NOW sub-block to the corresponding values in the PAR_BASE sub-block (see page 8). These differences are used to calculate STAT_SP.adapt. If adaption quality monitoring is disabled (MON_CFG.adapt = 0), STAT_SP.adapt = 0.

Specifically, if STAT_SP.adapt = 1, QM_ADAPT = 255. See QM_ADAPT on page 36 for more information.

STAT_SP.vtop indicates that the instantaneous AB output frequency, *fab*, has exceeded the limit set in register AB_VTOP (see page 20). This is not a fatal fault, but AB output frequency is limited in this condition and the position or angle indicated by the AB outputs no longer follows the sensor input. If FLT_EN.vwarn = 1, STAT_SP.vtop is "sticky" and stays activated even if the AB output frequency is subsequently reduced. See FLT_EN on page 28 for information on FLT_EN.vwarn.

STAT_SP.ospeed indicates that the instantaneous input frequency has exceeded $fcore/512$. This is a warning condition, not a fatal fault and the iC-TW8 continues to operate correctly. If FLT_EN.vwarn = 1, STAT_SP.opeed is "sticky" and stays activated even if the input frequency is subsequently reduced. See FLT_EN on page 28 for information on FLT_EN.vwarn.

STAT_SP.fatal indicates that the instantaneous input frequency has exceeded $fcore/256$. This is a fatal fault condition that results in erroneous AB outputs. If FLT_EN.vfatal = 1, STAT_SP.fatal is "sticky" and stays activated even if the input frequency is subsequently reduced. See FLT_EN on page 28 for information on FLT_EN.vfatal.

STAT_SP.lag indicates that the AB output position lags more than 7.9 input cycles behind the sensor position due to excessive filtering (see MAIN_FLTR on page 15). This is a fatal fault condition that results in erroneous AB outputs. If FLT_EN.vfatal = 1, STAT_SP.lag is "sticky" and stays activated even if the AB output frequency is subsequently reduced. See FLT_EN on page 28 for information on FLT_EN.vfatal.

STAT_SP.ab indicates that the AB output position lags too far behind the sensor position due to continued operation above the AB output frequency limit set in register AB_VTOP (see page 20). This is a fatal fault condition that results in erroneous AB outputs. If FLT_EN.vfatal = 1, STAT_SP.ab is "sticky" and stays activated even if the AB output frequency is subsequently reduced. See FLT_EN on page 28 for information on FLT_EN.vfatal.

STAT_SP.1wire indicates that a timeout fault on the 1-wire serial port (see page 6) has occurred and that the previous command or memory access was not performed. STAT_SP.1wire is "sticky" and stays activated until explicitly cleared by writing 0x00 to the STAT_SP register or by a device restart.

STAT_SP.xtal indicates that the iC-TW8 is configured to use an external crystal (see MAIN_CLOCK on page 17) but no external frequency is present. In this condition, the iC-TW8 reverts to using its internal oscillator for clocking. When or if the external frequency is subsequently applied, STAT-SP.xtal is cleared and the iC-TW8 again uses the external frequency for clocking.

STAT_EE (EEPROM Status Register)

The STAT_EE register contains bits that indicate the status of the external EEPROM (see page 10). It can be read to determine the cause of an EEPROM fault. STAT_EE register bits are "sticky" so that once activated, they stay activated even if the specific condition is subsequently resolved. Active STAT_EE bits can be cleared by writing 0x00 to the STAT_EE register.

STAT_EE (Word 0x809C)		
Bit	**Name**	**Description**
0	timeout	EEPROM communication timeout
1	stuck	EEPROM hardware fault
2	id	Wrong EEPROM ID
3	cfg	CFG block checksum error
4	par0	PAR Block 0 checksum error
5	par1	PAR Block 1 checksum error
6	base	PAR_BASE block checksum error
7	lut	LUT block checksum error
8	mode	Wrong configuration mode
9 – 15		Reserved

STAT_EE.timeout indicates that the external EEPROM did not respond to a read or write access from the iC-TW8. The most likely cause of this condition is a missing EEPROM.

STAT_EE.stuck indicates that the iC-TW8's SCL output (pin 27) or SDA input/output (pin 28) is stuck low.

STAT_EE.id indicates that the identification word read from the external EEPROM is not correct, most likely due to an uninitialized EEPROM. The Write All command (see page 33) can be used to initialize the EEPROM.

STAT_EE.cfg indicates that the CFG block stored in the EEPROM has a checksum error.

STAT_EE.par0 indicates that PAR block 0 stored in the EEPROM has a checksum error.

STAT_EE.par1 indicates that PAR block 1 stored in the EEPROM has a checksum error.

STAT_EE.base indicates that the PAR_BASE block stored in the EEPROM has a checksum error.

STAT_EE.lut indicates that LUT block stored in the EEPROM has a checksum error.

STAT_EE.mode indicates that the external EEPROM has an invalid configuration.

ADC_SIN, ADC_COS

ADC_SIN and ADC_COS are read-only registers containing recent ADC (Analog to Digital Converter) values for the Sin and Cos sensor input channels respectively (see page 30). They are signed 14-bit numbers in 2's complement format.

ADC_SIN (Word 0x80A4) ADC_COS (Word 0x80A6)	
Value	**Description**
±8192	Current ADC value

The actual ADC input voltages, *adcs* and *adcc*, are calculated as

$$adcs[\mathrm{V}] = \frac{\mathrm{ADC\_SIN}}{4096}$$

$$adcc[\mathrm{V}] = \frac{\mathrm{ADC\_COS}}{4096}$$

The effective Sin and Cos input signal levels, *sin* and *cos*, can be calculated as

$$sin[\mathrm{V}] = \frac{\frac{\mathrm{ADC\_SIN}}{4096} - \frac{\mathrm{AOFFS}}{10}}{10^{\frac{3\cdot\mathrm{AGAIN}}{20}}}$$

$$cos[\mathrm{V}] = \frac{\frac{\mathrm{ADC\_COS}}{4096} - \frac{\mathrm{AOFFC}}{10}}{10^{\frac{3\cdot\mathrm{AGAIN}}{20}}}$$

Note that ADC_SIN and ADC_COS are not updated in real time with every ADC sample, but only as other more important tasks permit. Thus they should be used only as a rough indication of proper ADC function during operation.

ADC_AMP (Signal Amplitude)

ADC_AMP is a read-only register containing the vector amplitude of the current ADC (Analog to Digital Converter) values for the Sin and Cos sensor input channels respectively (see page 30).

ADC_AMP (Byte 0x80A8)	
Value	**Description**
0 – 255	$\frac{\sqrt{\text{ADC\_SIN}^2 + \text{ADC\_COS}^2}}{32}$

The actual ADC vector signal output value, *adcamp*, is calculated as

$$adcamp = 32 \cdot \text{ADC\_AMP}$$

The effective analog ADC input vector signal amplitude, *amp*, is calculated as

$$amp[\text{V}] = \frac{\text{ADC\_AMP}}{128 \cdot 10^{\frac{3 \cdot \text{AGAIN}}{20}}}$$

QM_ADC (ADC Quality Monitor)

QM_ADC is a read-only register containing a value inversely proportional to the input signal amplitude between the ADC quality threshold and ADC fault level (see MON_ADC on page 24). It is used to determine the quality level signal on the STATUS output (pin 19).

QM_ADC (Byte 0x80AE)	
Value	**Description**
0 – 255	ADC quality monitor level

If MON_CFG.adc = 0 (see page 23), ADC quality monitoring is disabled and QM_ADC = 0.

If MON_CFG.adc = 1 and ADC_AMP ≥ *adcth* (see page 24), QM_ADC = 0.

If MON_CFG.adc = 1 and *adcflt* < ADC_AMP < *adcth* (see page 24), then

$$\text{QM\_ADC} = 256 \cdot \frac{adcth - \text{ADC\_AMP}}{adclim}$$

If MON_CFG.adc = 1 and ADC_AMP ≤ *adcflt* (see page 24), QM_ADC = 255.

QM_ADAPT (Adaption Quality Monitor)

QM_ADAPT is a read-only register containing a value proportional to the change in parameter values due to auto-adaption between the adaption quality thresholds and adaption fault levels. It is used to determine the quality level signal on the STATUS output (pin 19) and the adaption fault status (STAT_SP.adapt).

QM_ADAPT (Byte 0x80AF)	
Value	**Description**
0 – 255	Adaption quality monitor level

If MON_CFG.adapt = 0 (see page 23), adaption quality monitoring is disabled and QM_ADAPT = 0.

If MON_CFG.adapt = 1, QM_ADAPT is calculated as the sum of four adaption monitor values, one each for Sin channel digital offset (*moffs*), Cos channel digital offset (*moffc*), digital gain (*mgain*), and digital phase (*mphase*). Specifically,

$$\text{QM\_ADAPT} = moffs + moffc + mgain + mphase$$

If QM_ADAPT reaches 255, STAT_SP.adapt and FLT_STAT.adapt are set.

The individual adaption monitor values are calculated as described below.

If $|\text{DOFFS} - \text{PAR\_BASE\_DOFFS}| < offqth$ (see page 25), $moffs = 0$.

If $offqth \leq |\text{DOFFS} - \text{PAR\_BASE\_DOFFS}| < offflt$ (see page 25), then

$$moffs = 256 \cdot \frac{|\text{DOFFS} - \text{PAR\_BASE\_DOFFS}| - offqth}{offlim}$$

If $|\text{DOFFS} - \text{PAR\_BASE\_DOFFS}| \geq offflt$ (see page 25), then $moffs = 255$.

If $|\mathrm{DOFFC} - \mathrm{PAR\_BASE\_DOFFC}| < \mathit{offqth}$ (see page 25), $\mathit{moffc} = 0$.

If $\mathit{offqth} \le |\mathrm{DOFFC} - \mathrm{PAR\_BASE\_DOFFC}| < \mathit{offflt}$ (see page 25), then

$$\mathit{moffc} = 256 \cdot \frac{|\mathrm{DOFFC} - \mathrm{PAR\_BASE\_DOFFC}| - \mathit{offqth}}{\mathit{offlim}}$$

If $|\mathrm{DOFFC} - \mathrm{PAR\_BASE\_DOFFC}| \ge \mathit{offflt}$ (see page 25), then $\mathit{moffc} = 255$.

If $|\mathrm{DGAIN} - \mathrm{PAR\_BASE\_DGAIN}| < \mathit{gainqth}$ (see page 25), $\mathit{mgain} = 0$.

If $\mathit{gainqth} \le |\mathrm{DGAIN} - \mathrm{PAR\_BASE\_DGAIN}| < \mathit{gainflt}$ (see page 25), then

$$\mathit{mgain} = 256 \cdot \frac{|\mathrm{DGAIN} - \mathrm{PAR\_BASE\_DGAIN}| - \mathit{gainqth}}{\mathit{gainlim}}$$

If $|\mathrm{DGAIN} - \mathrm{PAR\_BASE\_DGAIN}| \ge \mathit{gainflt}$ (see page 25), then $\mathit{mgain} = 255$.

If $|\mathrm{DPH} - \mathrm{PAR\_BASE\_DPH}| < \mathit{phqth}$ (see page 26), $\mathit{mphase} = 0$.

If $\mathit{phqth} \le |\mathrm{DPH} - \mathrm{PAR\_BASE\_DPH}| < \mathit{phflt}$ (see page 26), then

$$\mathit{mphase} = 256 \cdot \frac{|\mathrm{DPH} - \mathrm{PAR\_BASE\_DPH}| - \mathit{phqth}}{\mathit{phlim}}$$

If $|\mathrm{DPH} - \mathrm{PAR\_BASE\_DPH}| \ge \mathit{phflt}$ (see page 26), then $\mathit{mphase} = 255$.

RB Block Registers

The RB block provides access to internal register banks of the iC-TW8. These register banks are used to invoke special test modes. The RB block must be accessed as words at an even byte address.

RB_TEST1

The RB_TEST1 Register is used to unlock the EEPROM and enable the clock test and Z test modes.

RB_TEST1 (Word 0xA008)		
Bit	**Name**	**Description**
0 – 2		Reserved
3	adctest	Enable clock test mode
4		Reserved
5	ztest	Enable Z test mode
6 – 8		Reserved
9	wpunlock	Unlock EEPROM
10 – 15		Reserved

Setting RB_TEST1.adctest enables clock test mode to allow tuning the frequency of the internal oscillator as explained in the Serial Configuration manual.

Setting RB_TEST1.ztest enables z test mode to allow calibration of the Z signal path as explained in the Serial Configuration manual.

Setting RB_TEST1.wpunlock unlocks an external EEPROM whose WP input is connected to the iC-TW8's WP output (pin 29) as explained on page 12.

Revision History

Date	Notes	Pages affected
Mar 20, 2012	First edition	All
Apr 14, 2012	*adclim* formula was wrong. Other changes for consistency and improvement.	24
April 19, 2012	Fixed MON_CFG, MON_ADC, MON_OFF, MON_GAIN, and MON_PHASE descriptions. Fixed FLT_EN details.	23 – 25 28
Sept 10, 2012	Correction of command byte re. 32-bit position read AB_CFG0.zpol descriptions were wrong.	4 18
Sept 21, 2012	Corrected ADC_AMP formulas.	35
Oct 9, 2012	Fixed formula font.	Many
Nov 22, 2012	Formal corrections of header layout; update of disclaimer	All
Jan 9, 2013	Add RB Block registers descriptions. Clarified Z output synchronization when using post-AB divider or non-integer interpolation factor. Corrected adaption correction formulas. Clarified adaption quality monitor operation and corrected adaption monitor formulas to use PAR_BASE instead of BAR_BAK. Clarified adaption fault behavior. Removed digital gain dB formulas.	37 18 – 19 22 – 23 25 – 26, 36 – 37 27, 34, 36 31
Dec 18, 2013	Added 1-Wire Command Byte table. Clarified operation of AB_VTOP and minimum edge separation when using the post-AB divider. Extended range of limit and threshold values when MON_CFG.adapt = 0. Corrected register addresses for AGAIN, AOFFS, and AOFFC. Command Register value of 0x01 is reserved. ADC_SIN and ADC_COS are not real-time.	7 20 25 – 26 30 33 35
Jul 7, 2016	SPI values msb first; position resolution determined by *inter*. 32-bit position updated at *fadc*. 1-Wire port not for general communication, Figs 5 and 6 updated. FLT_STAT not stored in EEPROM. Added MAIN_CLOCK.xforce bit and description. *inter* ≤ 1024 in PWM output mode. MAIN_Z.th has 30mV steps. Checksum errors in both PAR block 0 *and* 1 to activate FLT_STAT.ee. 1-Wire port inoperable during Restart command. STAT_SP.ospeed and STAT_SP.fatal refer to input frequency. 0 – 12 range for adaption monitor limit and threshold values regardless of MON_CFG.adapt value.	3, 5, 9 4, 9 6 13 13, 17 15, 16 18 27 33 34 25, 26
Aug 3, 2016	Removed "Preliminary".	All
Aug 16, 2016	MAIN_Z.reset limited to values of 0 or 1.	13, 18
Aug 9, 2017	Corrected formula for MON_CFG.adc. Corrected formulas for STAT_SP.adcof and STAT_SP.adcuf.	23 33, 34

Oct 19, 2017	Removed reference to linear encoder design tool. Added information regarding using the 32-bit position read to implement a multicycle counter. Enhanced AB_CFG0 register description to clarify that apol and bpol also reverse the AB counting direction and determine the AB startup states in relative startup mode (AB_CFG.startup = 0). Changed AB_CFG0.dir descriptions from "A leads B" to "normal" and from "B leads A" to "inverted."	2 5 18, 19 18