iND80205 – “Kamcho”

indie’s highly integrated M0 based microcontroller

11/04/2015 **Preliminary Data sheet**

1 TABLE OF CONTENT

2 LIST OF TABLES

3 LIST OF FIGURES

4 GENERAL DESCRIPTION

Kamcho integrates an ARM Cortex-M0 low cost 32-bit microcontroller containing 160kB of flash program memory and 8kB of SRAM. It implements several general-purpose peripherals. Its main features are:

CPU Architecture:

- ARM Cortex-M0 processor running at 12MHz (Internal RC), 32.768kHz RTC, or 10kHz (Internal auxiliary RC)
- System Tick Timer (SysTick – 24 bits, interruptible)
- Serial Wire Debugger
- Built-in Nested Vectored Interrupt Controller (NVIC)
- Programmable Watch-Dog Timer

Memory:

- 160kByte of Flash Program Memory
- 8kByte of SRAM
- Self-Programming

Peripherals:

- 39 General purpose I/O ports, 8 of which has LED current sink capability
- An ADC (8-bit), total of 41 input channels, with selectable input references and input gain block
- Temperature Sensor
- Low Battery Detection
- 2 x PWM (12-bit)
- Wake-up Timer
- 32.786kHz RTC; 10kHz RC oscillator; 12MHz, 1% RC oscillator
- UART Interface
- SPI Interface
- I2C Interface
- 500mA IR LED driver

Package:

- 7x7 48 pin QFN package

4.1 Block Diagram

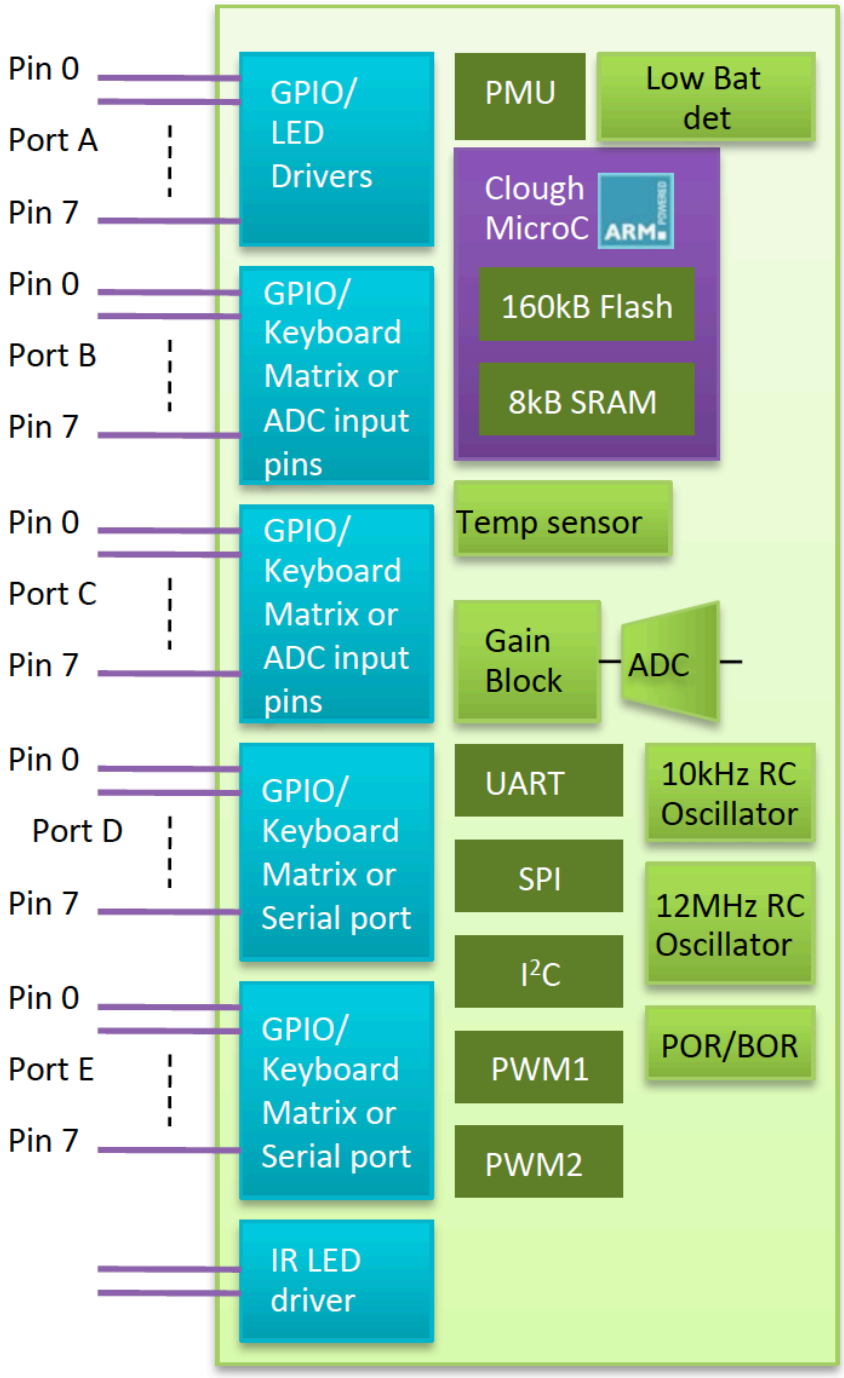


Figure 1 Kamcho block diagram

5 PINOUT

5.1 Kamcho 7x7 QFN pinout

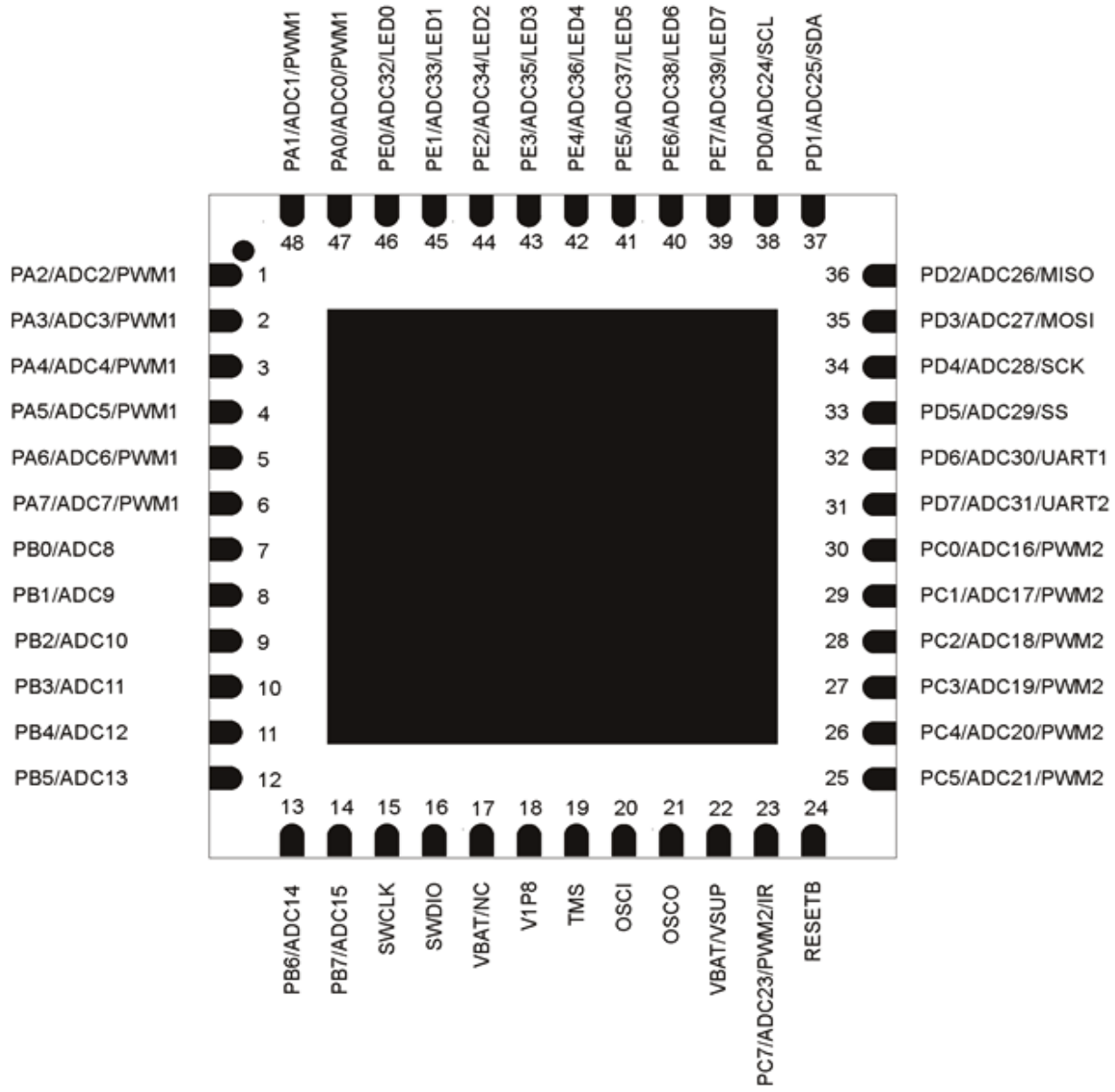


Figure 2 Kamcho Pinout Diagram (Top View)

5.2 Pin Description

Kamcho's pin list is defined below. In subsequent chapters the functionality of individual pins will be explained in details.

Table 1 ind81205 functional and supply pin list

#	Pin Name	Type	Direction	Description
1	PA2/ADC2/PWM1	GPIO		general purpose I/O with ADC input and PWM output
2	PA3/ADC3/PWM1	GPIO		general purpose I/O with ADC input and PWM output
3	PA4/ADC4/PWM1	GPIO		general purpose I/O with ADC input and PWM output
4	PA5/ADC5/PWM1	GPIO		general purpose I/O with ADC input and PWM output
5	PA6/ADC6/PWM1	GPIO		general purpose I/O with ADC input and PWM output
6	PA7/ADC7/PWM1	GPIO		general purpose I/O with ADC input and PWM output
7	PB0/ADC8	GPIO		general purpose I/O with ADC input
8	PB1/ADC9	GPIO		general purpose I/O with ADC input
9	PB2/ADC10	GPIO		general purpose I/O with ADC input
10	PB3/ADC11	GPIO		general purpose I/O with ADC input
11	PB4/ADC12	GPIO		general purpose I/O with ADC input
12	PB5/ADC13	GPIO		general purpose I/O with ADC input
13	PB6/ADC14	GPIO		general purpose I/O with ADC input
14	PB7/ADC15	GPIO		general purpose I/O with ADC input
15	SWCLK	Serial Wire		Debug Tool Serial Wire Clock
16	SWDIO	Serial Wire		Debug Tool Serial Wire Data I/O
17	VBAT/NC	Supply		Battery supply input or no connect when using internal regulator (1nF maximum load for internal regulator)
18	V1P8	other		1.8V regulator decoupling pin
19	TMS	Input		JTAG test mode select
20	OSCI	other		RTC crystal pin
21	OSCO	other		RTC crystal pin
22	VBAT/VSUP	Supply		Battery supply input or input to internal regulator when using 3.3V nominal supply
23	PC7/ADC23/PWM2/IR	GPIO		general purpose I/O with ADC input, PWM output and IR driver output

#	Pin Name	Type	Direction	Description
24	RESETB	Input		Active low reset signal
25	PC5/ADC21/PWM2	GPIO		general purpose I/O with ADC input and PWM output
26	PC4/ADC20/PWM2	GPIO		general purpose I/O with ADC input and PWM output
27	PC3/ADC19/PWM2	GPIO		general purpose I/O with ADC input and PWM output
28	PC2/ADC18/PWM2	GPIO		general purpose I/O with ADC input and PWM output
29	PC1/ADC17/PWM2	GPIO		general purpose I/O with ADC input and PWM output
30	PC0/ADC16/PWM2	GPIO		general purpose I/O with ADC input and PWM output
31	PD7/ADC31/UART2	GPIO		general purpose I/O with ADC input, UART-TXD or UART-RXD
32	PD6/ADC30/UART1	GPIO		general purpose I/O with ADC input, UART-TXD or UART-RXD
33	PD5/ADC29/SS	GPIO		general purpose I/O with ADC input, SPI Slave Select
34	PD4/ADC28/SCK	GPIO		general purpose I/O with ADC input, SPI Clock
35	PD3/ADC27/MOSI	GPIO		general purpose I/O with ADC input, SPI Master Out/Slave In
36	PD2/ADC26/MISO	GPIO		general purpose I/O with ADC input, SPI Master In/Slave Out
37	PD1/ADC25/SDA	GPIO		general purpose I/O with ADC input, I2C Data
38	PD0/ADC24/SCL	GPIO		general purpose I/O with ADC input, I2C Clock
39	PE7/ADC39/LED7	GPIO		general purpose I/O with ADC input, LED driver
40	PE6/ADC38/LED6	GPIO		general purpose I/O with ADC input, LED driver
41	PE5/ADC37/LED5	GPIO		general purpose I/O with ADC input, LED driver
42	PE4/ADC36/LED4	GPIO		general purpose I/O with ADC input, LED driver
43	PE3/ADC35/LED3/TCK	GPIO		general purpose I/O with ADC input, LED driver, JTAG Clock
44	PE2/ADC34/LED2/TDI	GPIO		general purpose I/O with ADC input, LED driver, JTAG Data In
45	PE1/ADC33/LED1/TDO	GPIO		general purpose I/O with ADC input, LED driver, JTAG Data Out
46	PE0/ADC32/LED0	GPIO		general purpose I/O with ADC input, LED driver
47	PA0/ADC0/PWM1	GPIO		general purpose I/O with ADC input and PWM output
48	PA1/ADC1/PWM1	GPIO		general purpose I/O with ADC input and PWM output
GND	Ground Pad	Supply		Ground slug on QFN package must be soldered to ground

6 Absolute Maximum Ratings

Absolute maximum ratings are defined in the following table. The operation of the device above these conditions may cause lasting damage and is not recommended.

Table 2 Absolute maximum Ratings

Name	Comments	Min.	Typ.	Max.	Unit
VBAT/VSUP voltage	instantaneous, no damage	-0.3		+4	V
GPIO voltage	configured as input, no damage, VBAT referenced	-0.3		VBAT+0.3	V
GPIO voltage	configured as input, no damage, VBAT referenced	-0.3		VBAT+0.3	V
Analog voltage	XTAL, VSUP no damage	-0.3		VBAT+0.3	V
Operating Temp.	de-rated performance, full functionality	-40		+85	°C
HBM (all pins)	to other pin in group or ground, soldered to application board	-2		2	kV
MM (all pins)	configured as input, no damage	-200		200	V

7 Recommended Operating Conditions

Table 3 Recommended Operating Conditions

name	Conditions	min	typ	max	unit
VBAT voltage		2.3	3	3.2	V
VSUP Voltage		2.9	3.3	3.6	V
Operating Temp.		-40	25	85	°C

8 CURRENT CONSUMPTION

Table 4 Current Consumption					
Name	**Conditions**	**Min**	**Typ**	**Max**	**Unit**
Deep Sleep Mode	Aux Clock running, CPU disabled, 1.8V regulator disabled		1		μA
Sleep Mode	RTC running, CPU running, 1.8V regulator enabled for full speed operation		500		μA
Normal Operation	CPU running, Main RC as clock, CPU clock /8		1400		μA
Full Speed Operation	CPU running, Main RC as clock, CPU clock /1		3000		μA

9 REGISTER CONVENTION

Several registers will be defined and explained throughout this document. The general format of the description of the registers is as follows:

Name of the Register		Starting Address (Hex)			Reset or Default Value (Hex)		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit_Name	Bit_Name	Bit_Name	Bit_Name	Bit_Name	Bit_Name	Bit_Name	Bit_Name
MSB							LSB

Where R/W is the read and write permissions of the specific bit. An example:

TMR0REG		0x50020000			0x00000000		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
T7	T6	T5	T4	T3	T2	T1	T0
T15	T14	T13	T12	T11	T10	T9	T8
T23	T22	T21	T20	T19	T18	T17	T16
T31	T30	T29	T28	T27	T26	T25	T24
MSB							LSB
Bit31-0 **T[31:0]**: Timer Register initial value register.							

The name of this register is TMR0REG (Timer 0 Register). It is a 32-bit register, located at address 0x50020000, 0x50020001, 0x5002002 and 0x50020003. The first row of the data (T[7:0) corresponds to address 0x50020000 and the fourth row of the data (T[31:24]) corresponds to address 0x50020003.

10 MEMORY MAP

The following tables show the memory partitioning.

Table 5 System Memory Map

Address	Memory	Description	Reference
0x00000000 - 0x00027FFF	Flash	160kByte Flash Memory	N/A
0x20000000 - 0x20001FFF	SRAM	8kByte SRAM	N/A
0x50000000 - 0x5000007F	Peripheral	128Byte peripheral fast access	Table 7
0x50000080 - 0x50000085	Peripheral	6Byte Block Transfer control	N/A
0x50010000 - 0x5001FFFF	Peripheral	64kByte peripheral slow access	Table 6
0x50020000 - 0x5002001F	Peripheral	32Byte timer control	N/A
0x50020020 - 0x50020047	Flash	40Byte Flash program/erase control	N/A
0xE0000000 - 0xE00FFFFF	Private peripheral bus	ARM peripherals	N/A
0xF0000000 - 0xF0001FFF	System ROM tables	ARM core IDs	N/A

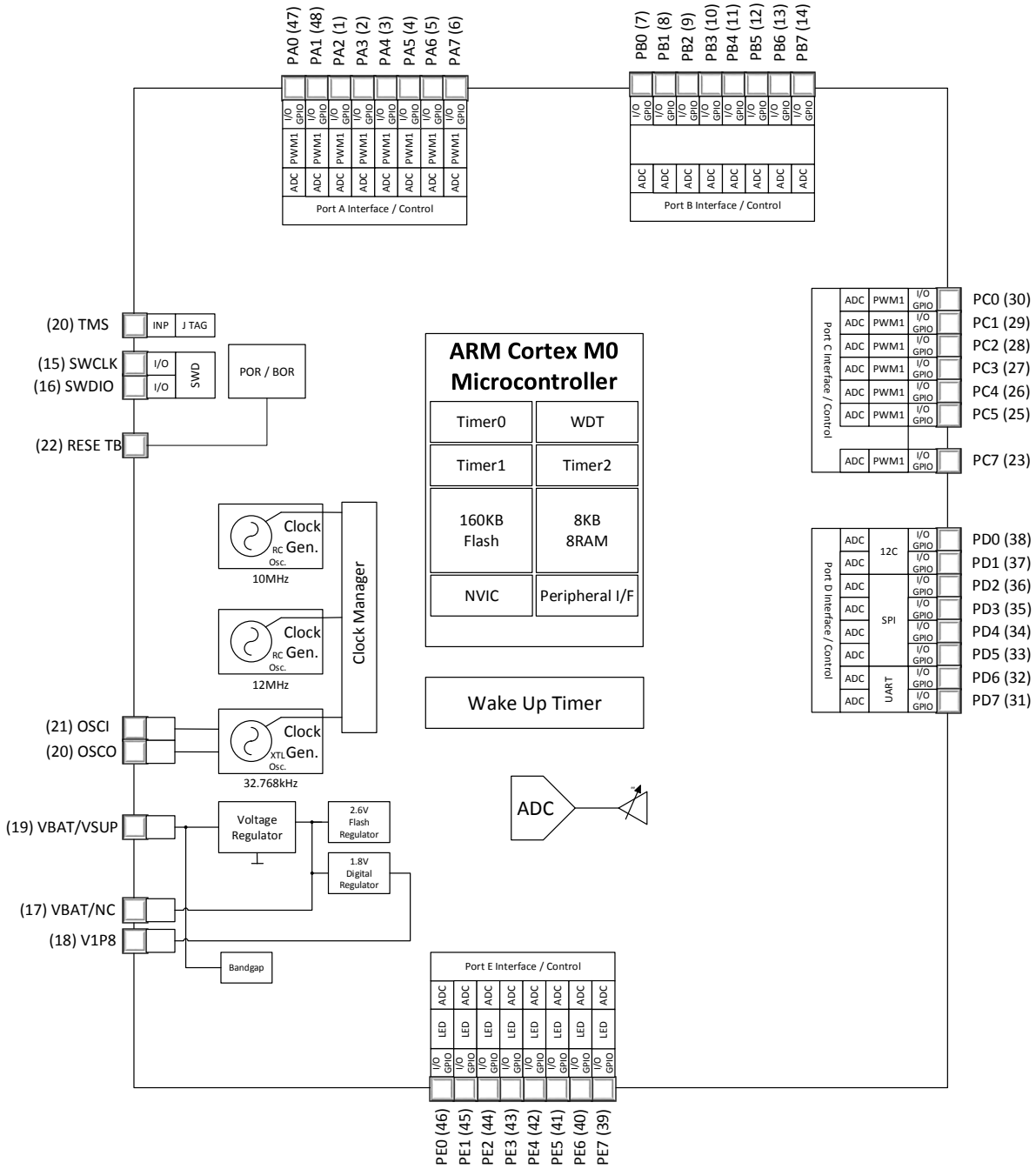
Table 6 Peripheral Slow Access Memory Map

Address	Peripheral	Description	Reference
0x50010000 - 0x500104FF	GPIO	GPIO bit control	
0x50018004 - 0x50018013	ADC	ADC and miscellaneous control	

Table 7 Peripheral Fast Access Memory Map			
Address	Peripheral	Description	Reference
0x50000000 - 0x50000005	PMU	Power management unit control	
0x50000008 - 0x5000000F	I2C	I2C bus control	
0x50000010 - 0x50000017	UART	UART control	
0x5000001C - 0x5000001F	SPI	SPI control	
0x50000048 - 0x5000004F	PWM	Pulse width modulator control	
0x50000050 - 0x5000005A	ADC	ADC and shock sensor control	
0x50000060 - 0x5000007F	GPIO	GPIO Byte control	

11 FUNCTIONAL BLOCKS AND ELECTRICAL CHARACTERISTICS

11.1 System Bloc diagram



11.2 Microcontroller Subsystem

Kamcho includes an embedded microcontroller subsystem, which is based on the ARM Cortex M0 core. It includes a program flash memory of 160kBytes, and an SRAM of 8kBytes. It includes three 32-bit timers, plus a dedicated watchdog timer. Additionally, it includes a **N**ested **V**ector **I**nterrupt **C**ontroller (NVIC) to scheduled hardware interrupts, and a **W**akeup **I**nterrupt **C**ontroller (WIC), which enable the control of the various power modes. Further information can be obtained in the specification document [1] "***AyDeeKay_Core_160_8.pdf"***.

11.2.1 Timers (0,1, and 2)

Kamcho implements three identical timers: Timer0, Timer1 and Timer2. These timers use the system clock as clock source and once activated count up continuously. They start from the value initially loaded into the counting register (32-bit) and, if enabled, generate an interrupt upon rolling over (0xFFFFFFFF → 0x00000000).

11.2.1.1 Timers Registers

There are two basic registers associated with each of three timers:

Register 1		**32-bit Timer0 initial value register**					
TMR0REG		0x50020000			0x00000000		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
T7	T6	T5	T4	T3	T2	T1	T0
T15	T14	T13	T12	T11	T10	T9	T8
T23	T22	T21	T20	T19	T18	T17	T16
T31	T30	T29	T28	T27	T26	T25	T24
MSB							LSB
Bit31-0 **T[31:0]**: Timer Register initial value register.							

Register 2 Timer0 Control register							
TMR0CTRL		0x50020004			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	TSTART
MSB							LSB
Bit0 **TSTART**: Timer enable bit. 0 = Timer not running 1 = Timer running							

Register 3 32-bit Timer1 initial value register							
TMR1REG		0x50020008			0x00000000		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
T7	T6	T5	T4	T3	T2	T1	T0
T15	T14	T13	T12	T11	T10	T9	T8
T23	T22	T21	T20	T19	T18	T17	T16
T31	T30	T29	T28	T27	T26	T25	T24
MSB							LSB
Bit31-0 **T[31:0]**: Timer Register initial value register.							

Register 4 Timer1 Control register							
TMR1CTRL		0x5002000C			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	TSTART
MSB							LSB
Bit0 **TSTART**: Timer enable bit. 0 = Timer not running 1 = Timer running							

Register 5 32-bit Timer2 initial value register							
TMR2REG		0x50020010			0x00000000		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
T7	T6	T5	T4	T3	T2	T1	T0
T15	T14	T13	T12	T11	T10	T9	T8
T23	T22	T21	T20	T19	T18	T17	T16
T31	T30	T29	T28	T27	T26	T25	T24
MSB							LSB
Bit31-0 **T[31:0]**: Timer Register initial value register.							

Register 6 Timer2 Control register							
TMR2CTRL		0x50020014			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	TSTART
MSB							LSB

Bit0 **TSTART**: Timer enable bit.

0 = Timer not running

1 = Timer running

11.2.1.2 Timer Operation

The operation of the timers is quite straightforward. Load the initial counter register, enable the timer and either check (polling mode) the current value of the counter register or enable the interrupt and process it inside the interrupt service routine.

Note: Inside the interrupt the application code must reload the timer counting register.

Code Example1: Enable Timer1 to count from 0xFFFF0000 and to generate interrupt:

```
TMR_Config( 1, TIMERON, 0xFFFF0000);  //Enable timer1 to count up from 0xFFFF0000
NVIC_EnableIRQ( TIMER1_IRQn );          //Enable Timer1 interrupt
void Timer1_Handler( void )
{
  *TMR1REG = 0xFFFF0000;                //Reload Register
  //**** From this point application code inside ISR****
}
```

11.2.2 Watch Dog Timer

Kamcho implements a WDT (**W**atch **D**og **T**imer) that can operate in one of two basic ways:
Interrupt Mode: In the event of a WDT rollover an interrupt will be generated.
Reset Mode: In the event of a WDT rollover the microcontroller will reset.

11.2.2.1 WDT Registers

The Watch Dog Timer implements two 32-bit registers:

Register 7		**Watch Dog Timer control register (32-bit)**					
WDTCTRL		0x50020018			0x0000000x		
Reserved	Reserved	Reserved	R/W	R/W	R/W	R/W	R/W
-	-	-	WDTPRES1	WDTPRES0	RSTFLAG	RESETEN	WDTEN
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
MSB							LSB

Bit4-3 **WDTPRES1: WDTPRES0**: WDT Prescaler:

00 : $t_{timout} = 2^{13}/f_{SystemClock}$

01 : $t_{timout} = 2^{19}/ f_{SystemClock}$

10 : $t_{timout} = 2^{22}/ f_{SystemClock}$

11 : $t_{timout} = 2^{32}/ f_{SystemClock}$

Bit2 **RSTFLAG**: Reset Flag. This flag is set by the system at the initialization if the initialization was caused by a reset triggered by the WDT. The bit can be de-asserted by the application.

Bit1 **RESETEN**: Reset enable. If enabled a WDT time-out will force the microcontroller to reset. This bit can be asserted but it cannot be de-asserted.

0 = WDT will generate an interrupt if enabled in NVIC by the application

1 = WDT will generate a reset

Bit0 **WDTEN**: WDT enable. This bit can be asserted but it cannot be de-asserted. It means that once the WDT is enabled it cannot be turned off until a Reset or Power-On Reset occurs.

For instance, a system running from a 30MHz Crystal with WDTPRES[4..3] = 10 will trigger the WDT after approximately 0.14seconds if not cleared properly and in time by the application.

Register 8 WDT Clear register (32-bit)							
WDTCLR		0x5002001C			0x0000000x		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
WCLR7	WCLR6	WCLR5	WCLR4	WCLR3	WCLR2	WCLR1	WCLR0
WCLR15	WCLR14	WCLR13	WCLR12	WCLR11	WCLR10	WCLR9	WCLR8
WCLR23	WCLR22	WCLR21	WCLR20	WCLR19	WCLR18	WCLR17	WCLR16
WCLR31	WCLR30	WCLR29	WCLR28	WCLR27	WCLR26	WCLR25	WCLR24
MSB							LSB

Bit31-0 **WCLR[31:0]**: Clear Register. To clear the WDT counting the following words must be written in this order and without any other instruction between then:

0x3C570001

0x007F4AD6

Warning: Programming WDTCLR with other values or in the wrong order will cause the watchdog to generate an interrupt or reset the system.

Example Code: Setting and clearing the WDT. (Interrupt mode with a time of 2^22)

```
WDT_Config(WDT_INT, WDT22);        //Enable WDT in interrupt mode (2^22 system clock cycles)
WDT_Clear();                       //Clear WDT
```

11.2.3 Interrupt Vectors

Kamcho implements an interrupt vector defined in the following tables:

Table 8 Interrupt Vectors			
Cortex M0 Specific Exceptions			
Name	Number	Comments	Required Interrupt Handler (Function)
HardFault_IRQn	-13	HardFault handler*	HardFault_Handler (void)
SVCall_IRQn	-5	Supervisory call*	
PendSV_IRQn	-2	Interrupt-driven request for system level service*	
SysTick_IRQn	-1	SysTick Timer interrupt	void SysTick_Handler(void)

Table 9 Interrupt Vectors			
Kamcho Specific Exceptions			
Name	Number	Comments	Required Interrupt Handler (Function)
BrownOut_IRQn	0	Brownout detection interrupt	void BrownOut_Handler (void)
ClkMon_IRQn	1	Clock monitor interrupt	void ClkMon_Handler (void)
PIN_IRQn	2	Pin change interrupt	void PIN_Handler (void)
RTC_FE_IRQn	3	RTC falling edge interrupt	void RTCFE_Handler (void)
RTC_RE_IRQn	4	RTC rising edge interrupt	void RTCRE_Handler (void)
I2C_Collision_IRQn	5	I^2C Collision detection interrupt	void I2C_Collision_Handler (void)
I2C_IRQn	6	I^2C event interrupt	void I2C_Handler (void)
UART_IRQn	7	UART event interrupt	void UART_Handler (void)
SPI_IRQn	8	SPI event interrupt	void SPI_Handler (void)
IRQ9_IRQn to IRQ15_IRQn	9-15	Reserved	void Default_IRQ_Handler(void)
TIMER0_IRQn	16	Timer0 interrupt	void Timer0_Handler (void)
TIMER1_IRQn	17	Timer1 interrupt	void Timer1_Handler (void)
TIMER2_IRQn	18	Timer2 interrupt	void Timer2_Handler (void)
WATCHDOG_IRQn	19	Watchdog timer interrupt	void Watchdog_Handler (void)

*Note: For more information see *Cortex-M0 Devices – Generic Users Guide (ARM DUI 0497A (ID112109))*
at: http://infocenter.arm.com/help/topic/com.arm.doc.dui0497a/DUI0497A_cortex_m0_r0p0_generic_ug.pdf

11.3 UART

Kamcho includes a UART (**U**niversal **A**synchronous **R**eceiver **T**ransmitter) module. The main characteristics are defined below:

- Interrupt available for transmission, reception and error events
- Reception timeout timer
- Programmable break reception and transmission
- Programmable parity with “sticky” parity option
- Selectable number of bits from 5 to 8
- Selectable number of stop-bits: 1, 1 ½, 2
- Programmable loop-back
- Swappable TXD and RXD (PD[6] and PD[7])
- Transmitter Polarity selection

11.3.1 UART Operation

The UART protocol requires two wires (UTXD and URXD). PD[7] and PD[6] are configured as UTXD and URXD when the MDUART bit is set in the pin configuration register, PCONF. In order to use the UART, the following steps must be followed:

- Step 1: Select the pins position (normal or swapped) of the interface and also its polarity. The normal position (not swapped) is TX=PD[7] and RX = PD[6].

Code Example: Selecting UART with normal polarity and swapped: (UART pins swapped: TX=PD[6] and RX = PD[7])

UART_Setup(UARTSWAP_EN, UARTPOL_NORMAL, UART_MODE_EN);

- Step 2: Define the following parameters:
    - Loop back: Used mainly in tests, internally connects the output to the input.
    - Break enable: Pulls the output down while asserted, raising the output once de-asserted.
    - Sticky parity: Forces the parity to stay stable in one direction.
    - Even/Odd parity selection and enable: Selection and enable of Even or Odd parity bits.
    - Number of stop bits: Selection of 1 (default), 1½ (5-bit communications only) or 2 stop bits.
    - Data size in bits (5,6,7,8): Selection of the number of bits used in the communication

Code example: Setting the above parameters: (no loop-back, no break signal, no sticky parity, no parity, one stop-bit, 8-bit communications)

```
UART_Ctrl(UART_LBDIS, UART_BREAKDIS, UART_STPDIS,UART_ODDEN, UART_PARDIS, \
                UART_10STOP, UART_8BITS);
```

Define the baud rate. The baud rate is calculated as follows: (UARTDIV is a 16-bit register)

$$Baud = \frac{Fclk}{16 * (UARTDIV + 1)}$$

Assuming a 4MHz system clock the following table provides some register values, baud rates and related errors:

Table 10 UART baud rates, divider values and errors

Baud	UARTDIV	Real Baud	Error (%)
300	832	300.1	0.04
600	416	599.5	0.08
1200	207	1202	0.16
2400	103	2404	0.16
4800	51	4808	0.16
9600	25	9615	0.16
19200	12	19231	0.16

Code Example: Setting the UART to operate at 9600 baud:

```
UART_BaudRateDivider(22);
```

- Step 3: Enable the UART and its interrupt: The UART may generate an interrupt for events related to:
  - Transmission completed.
  - Reception: Timeout of ~40 bit-times without reception, and data received.
  - Errors detected: Framing error, parity error, and overrun error.
  - Break signal detected (received).

Code example: Enabling the UART with no timeout interrupts, errors, transmission and reception interrupts with interrupts once 1 byte is received in the FIFO.

```
//Timeout interrupt disabled, Error enabled, TX enab69
, RX enabled
      UART_Interrupt_Control(UART_TOUTDIS,UART_ERREN,UART_TXEN,UART_RXEN);
//UART enabled
      UART_Ctrl1( UARTEN);

//Enabling interrupt from UART at the microcontroller

      NVIC_EnableIRQ(UART_IRQn);         //Enable UART interrupt
```

Processing of the UART interrupt:

```
void UART_Handler( void )           // IRQ 8 UART
{
      switch(UART_Interrupt_Status())
      {
            case  UART_ERROR:
                  //Process reception error here
                  SWITCH(UART_CheckError())
                  {
                        case UART_FRAMING_ERROR: // Process this error here
                              break;
                        case UART_PARITY_ERROR: // Process this error here
                              break;
                        case UART_OVERRUN_ERROR: // Process this error here
                              break;
                        case 0: // No error, exit
                        default:
                              break;
                  }
                  break;
            case  UART_RXRDY:
                  //Process data received
                  mydata = *UARTDATA;     //Read data received by uart into mydata
                  break;
            case  UART_TIMEOUT:
                  //Process reception timeout
                  break;
            case  UART_TXDONE:
                  //Process transmission complete
                  break;
            case  UART_NOINT: //No int.
            default:
            //If no interrupt asserted or something else happened nothing to do
                        break;
      }
}
```

11.3.2 UART Registers

Table 11 UART Control Register Map				
Address	Register Name	Description	Reset Value	Reference
0x50000010	UARTDATA	UART Data Register	0x00	
0x50000011	UARTICTRL	UART Interrupt Control Register	0x00	
0x50000012	UARTLCTRL	UART Line Control Register	0x00	
0x50000013	UARTEN	UART Enable Register	0x00	
0x50000014	UARTLSTAT	UART Line Status	0x00	
0x50000016-7	UARTDIV	UART Baud Rate Divider	0x0000	

Register 9 UART Data Register							
UARTDATA		0x50000010			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
UARTD7	UARTD6	UARTD5	UARTD4	UARTD3	UARTD2	UARTD1	UARTD0
MSB							LSB
Bit7-0 **UARTD [7:0]:** UART data, both received and to be transmitted.							

Register 10 UART Interrupt Control Register							
UARTICTCL		0x50000011			0x00		
R	R	R	R	R/W	R/W	R/W	R/W
UISTTS3	UISTTS2	UISTTS1	UISTTS0	UTOUTIEN	URXERREN	UTXIEN	URXIEN
MSB							LSB

Bit7-4 **UISTTS [3:0]:** UART Interrupt status:

0001 = No Interrupt asserted

0010 = Transmission completed

0100 = Data received

0110 = Reception error

1100 = Reception timeout (~40 bit-time)

Bit3 **UTOUTIEN**: UART time-out interrupt enable bit:

0 = Time-out interrupt disabled

1 = Time-out interrupt enabled

Bit2 **URXERREN**: UART reception error interrupt enable bit:

0 = Reception error interrupt disabled

1 = Reception error interrupt enabled

Bit1 **UTXIEN**: UART transmission completed interrupt enable bit:

0 = Transmission completed interrupt disabled

1 = Transmission completed interrupt enabled

Bit0 **URXIEN**: UART reception interrupt enable bit:

0 = Reception interrupt disabled

1 = Reception interrupt enabled

Register 11 UART Line Control Register							
UARTLCTRL		0x50000012			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ULOOPEN	UBREAKEN	USTICKEN	UPARITY	UPAREN	USTOP	USTOP	USIZE
MSB							LSB

Bit7 **ULOOPEN**: UART loop back enable:

0 = UART loop back disabled

1 = UART loop back enabled

Bit6 **UBREAKEN:** UART break enable:

0 = UART break disabled

1 = UART break enabled

Bit5 **USTICKEN**: UART sticky parity enable bit:

0 = Sticky parity disabled

1 = Sticky parity enabled

Bit4 **UPARITY**: UART parity bit:

0 = Odd parity

1 = Even parity

Bit3 **UPAREN**: UART parity enable bit:

0 = Parity disabled

1 = Parity enabled

Bit2 **USTOP**: UART stop bit:

0 = One stop bit

1 = If a 5-bit transmission it selects1.5 stop bits, otherwise 2 stop bits (6, 7 and 8 bits)

Bit1-0 **USIZE**: UART transmission size:

00 = 5-bit data

01 = 6-bit data

10 = 7-bit data

11 = 8-bit data

Register 12 UART Enable Register							
UARTCTRL1		0x50000013			0x00		
Reserved	Reserved	Reserved	Reserved	R/W	Reserved	Reserved	Reserved
-	-	-	-	UARTEN	-	-	-
MSB							LSB
Bit3 **UARTEN**: UART enable: 0 = UART disabled 1 = UART enabled							

Register 13 UART Line Status Register							
UARTSTATUS		0x50000014			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
UERR	UTXEMPTY	UTXFFEMPTY	UBREAKINT	UFRMERR	UPRTYERR	UOVRUNERR	UDTRDY
MSB							LSB

Bit7 **UERR**: UART error:

0 = No error

1 = Error in UART

Bit6 **UTXEMPTY:** UART transmission empty:

0 = UART transmitter not empty

1 = UART transmitter empty

Bit5 **UTXFFEMPTY**: UART transmission FIFO empty:

0 = TX FIFO not empty

1 = TX FIFO empty

Bit4 **UBREAKINT**: UART break interrupt:

0 = No break interrupt

1 = Break interrupt

Bit3 **UFRMERR**: UART framing error:

0 = UART no framing error

1 = UART framing error

Bit2 **UPRTYERR**: UART parity error:

0 = No parity error

1 = Parity error

Bit1 **UOVRUNERR**: UART overrun error:

0 = No overrun error

1 = Overrun error

Bit0 **UDTRDY**: UART data ready:

0 = No data ready (reception)

1 = Data ready (reception)

Register 14 UART Baud Rate Divider Register (16-bit)							
UARTDIV		0x50000016			0x0000		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
UDIV7	UDIV6	UDIV5	UDIV4	UDIV3	UDIV2	UDIV1	UDIV0
UDIV15	UDIV14	UDIV13	UDIV12	UDIV11	UDIV10	UDIV9	UDIV8
MSB							LSB
Bit15-0: **UDIV [15:0]**: UART clock divider							

11.4 SPI Interface

The Serial Peripheral Interface (SPI) is a synchronous full-duplex serial interface. It communicates in master/slave mode where the master initiates the data transfer. In Kamcho, the SPI is implemented as a master. The module is compatible with an industry standard SPI interface. There are many references available, but a simple overview can be found at:

[2] http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

Kamcho SPI module's main features are defined below:

- Compatible with Industry Standard SPI interface
- Four bytes deep reception FIFO
- Four bytes deep transmission FIFO
- Interrupt upon events related to transmission, reception and error:
    - Write Collision
    - Transmission FIFO full and empty
    - Reception FIFO full and empty
- Expandable to more devices via using GPIOs and Software

The SPI protocol requires four wires (SCK, MISO, MOSI, and SS). The pins PD[5:2] are configured as the SPI bus when the MDSPI bit is set in the pin configuration register, PCONF. The following table describes how each pin is connected:

Table 12 SPI Interface Signals			
Name	**Pin Number**	**Pin Name**	**Comments**
MISO	36	PD2	Mater In Slave Out
MOSI	35	PD3	Master Out Slave In
SCLK	34	PD4	Serial Clock
SS	33	PD5	Slave Select

11.4.1 SPI Functionality

Only the master mode is implemented in Kamcho. Kamcho configures the clock frequency and generates the serial clock (SCK) for the interface. The data transfer is synchronous through SCK. The SPI is a full-duplex system; data is transmitted and received simultaneously. Kamcho sends the information to the slave device through the MOSI line and receives the data through MISO line. The CPOL and CPHA bits in the SPI control register determine when to sample the data.

When CPOL=0, the base value of the clock is logic '0'. In this case, if CPHA=0, data is captured on the rising edge of SCK and data is propagated on the falling edge of SCK. For CPHA=1, data is captured on the falling edge of SCK and data is propagated on the rising edge of SCK.

If CPOL=1, the base value of clock is logic '1'. In this case, if CPHA=0, data is captured on the falling edge of SCK and data is propagated on the rising edge of SCK. For CPHA=1, data is captured on the rising edge of SCK and data is propagated on the falling edge of SCK.

The timing diagram is shown below.

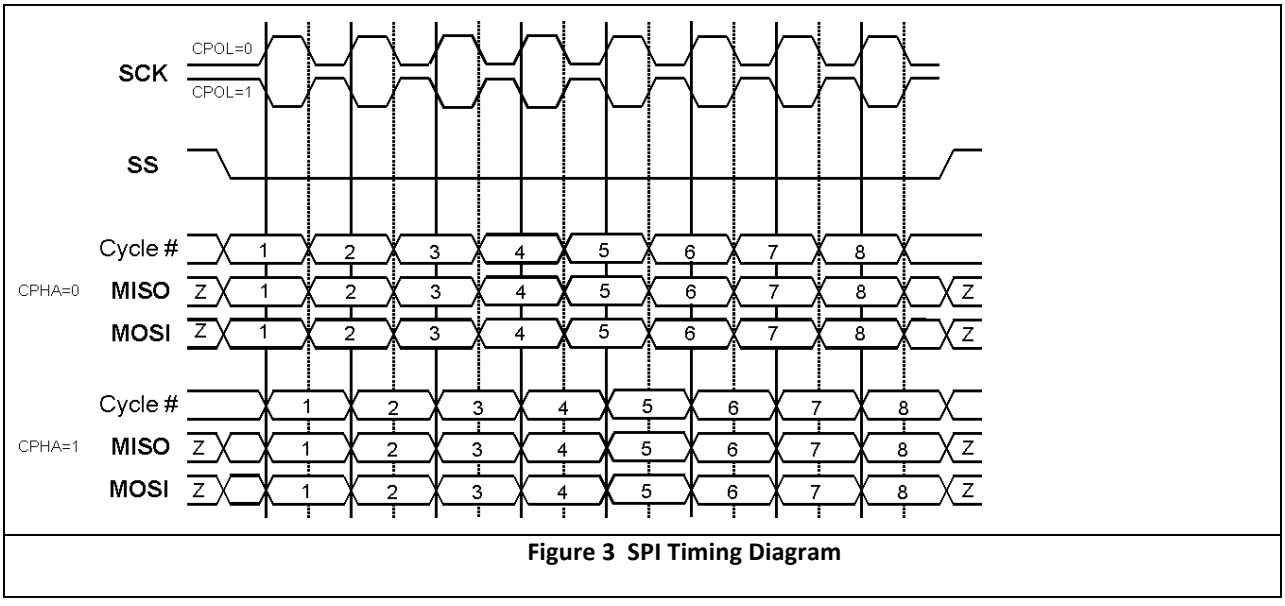


Figure 3 SPI Timing Diagram

After a desired configuration is set through configuration registers, a transfer is initiated by writing to the SPI Data Register (SPDR). The data is input into a 4-deep FIFO before it is transmitted. When the data is transmitted, the slave also transmits the data simultaneously for Kamcho to receive. The received data is stored in a separate 4-deep FIFO. The data is accessed by reading SPDR register.

To operate it properly the following steps must be performed:

- Step 1: Configure and enable the SPI: Select if the interrupt is enabled, set the polarity, phase and the clock divider:

Code Example: Enabling the SPI interface with interrupt enabled, clock polarity negative (base value = 0), phase1 (data captured on the clock's falling edge and propagated on the rising edge), and divider = 2.

```
SPI_Config(SPI_INT_EN,SPI_EN,SPI_CPOL_NEG,SPI_PHASE1,2);
```

- Step 2: Enable the interrupt (if required):

Code Example:

```
//Enabling interrupt from SPI at the microcontroller

NVIC_EnableIRQ(SPI_IRQn);          //Enable SPI interrupt
```

- Step 3: Process the Interrupt (if required) and detect the reason for the interrupt (error, transmission or reception related and act accordingly):

Code Example: Processing SPI interrupt:

```
void SPI_Handler( void )            // IRQ A SPI
{
      uint8_t status;
      if ( (status = SPI_ReadStatus() ) & SPI_INT_FLAG )
      {
            if (status == SPI_WCOL)
            {
            //Process write collision (data is written to the SPI data
            //register while a SPI data transfer is in progress)

            }
            if (status == SPI_TX_FIFO_FULL)
            {
            //Process Transmission FIFO full
            }
            if (status == SPI_TX_FIFO_EMPTY)
            {
            //Process end of transmission of data previously
            //in transmission FIFO
                  *SPIDATA = outdata[j++];
                  *SPIDATA = outdata[j++];
                  *SPIDATA = outdata[j++];
            }
            if (status == SPI_RX_FIFO_FULL)
            {
            //Process reception FIFO full, normally by reading
            //all bytes of data
                  for ( I = SPI_ReadRxFifoSize(); I > 0; i--)
                        mydata[i++] = *SPIDATA;
            //Other processing here
            }
            if (status == SPI_RX_FIFO_EMPTY)
            {
            //Process when no more information is available (received)
            }
      }
}
```

11.4.2 SPI Registers

The following registers are defined in the SPI interface:

Table 13 SPI Control Register Map				
Address	Register Name	Description	Reset Value	Reference
0x5000001C	SPCR	SPI Control Register	0x10	Register 15
0x5000001D	SPSR	SPI Status Register	0x00	Register 16
0x5000001E	SPDR	SPI Data Register	0x00	Register 17
0x5000001F	SPER	SPI Extension Register	0x00	Register 18

Register 15 SPI Control Register							
SPCR		0x5000001C			0x10		
R/W	Reserved	Reserved	R	R/W	R/W	R/W	R/W
SINTE	-	-	MSTR	CPOL	CPHA	SCKSTD1	SCKSTD0
MSB							LSB

Bit7 **SINTE**: SPI Interrupt enable

0 = Interrupt is disabled

1 = Interrupt is enabled

Bit4 **MSTR:** Master Mode Select Bit

SPI is always in master mode in Kamcho, and therefore, it is always set to logic '1'.

Bit3 **CPOL** SPI clock polarity

0 = The base value of the clock is zero

1 = The base value of the clock is one

Bit2 **CPHA**: SPI clock phase

0 = data is captured on clock transition from base and data is propagated on the clock transition to base

1 = data is captured on clock transition to base and data is propagated on the clock transition from base

Bit1-0 **SCKSTD[1:0]**: SPI standard clock divider selection

Please refer to SPER register for system clock

Register 16 SPI Status Register							
SPSR		0x5000001D			0x00		
R/W	R/W	Reserved	Reserved	R/W	R/W	R/W	R/W
SINTF	SWCOL	-	-	STXFF	STXFE	SRXFF	SRXFE
MSB							LSB

Bit7 **SINTF**: SPI interrupt flag

0 = Interrupt not asserted

1 = Interrupt asserted

Bit6 **SWCOL:** SPI write collision is set when the SPDR register is written to while the transmit FIFO is full

0 = No collision

1 = collision

Bit3 **STXFF**: SPI transmit FIFO full

0 = transmit FIFO not full

1 = transmit FIFO full

Bit2 **STXFE**: SPI transmit FIFO empty

0 = transmit FIFO not empty

1 = transmit FIFO empty

Bit1 **SRXFF**: SPI reception FIFO full

0 = reception FIFO not full

1 = reception FIFO full

Bit0 **SRXFE**: SPI reception FIFO empty

0 = reception FIFO not empty

1 = reception FIFO empty

Register 17 SPI Data Register							
SPDR		0x5000001E			0xXX		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
SPID7	SPID6	SPID5	SPID4	SPID3	SPID2	SPID1	SPID0
MSB							LSB
Bit7-0 **SPID[7:0]**: SPI data, used in both transmission and reception							

Register 18 SPI Extension Register							
SPER		0x5000001F			0x00		
R/W	R/W	Reserved	Reserved	R/W	R/W	Reserved	Reserved
SICNT1	SINCT0	-	-		SPE	SCKEXT1	SCKEXT0
MSB							LSB

Bit7-6 **SICNT[1:0]**: SPI Interrupt Counter Bits

00 = SINTF is set after every completed transfer

01 = SINTF is set after two completed transfers

10 = SINTF is set after three completed transfers

11 = SINTF is set after four completed transfers

Bit2 **SPE**: SPI Enable

0 = SPI module is disabled

1 = SPI module is enabled

Bit1-0 **SCKEXT[1:0]**: SPI extended clock divider

SCKSTD	SCKEXT	Result Clock Divider
00	00	= System Clock/2
01	00	= System Clock/4
10	00	= System Clock/16
11	00	= System Clock/32
00	01	= System Clock/8
01	01	= System Clock/64
10	01	= System Clock/128
11	01	= System Clock/256
00	10	= System Clock/512
01	10	= System Clock/1024
10	10	= System Clock/2048
11	10	= System Clock/4096
xx	11	= Reserved

11.5 I²C Interface

Kamcho includes an I²C interface. Its main characteristics are:

- Support for multi-master mode
- General call support
- 10-bit address
- Address masking

The I²C interface is a well-known interface and many references that describe its behavior are available. As an example:

http://en.wikipedia.org/wiki/I%C2%B2C

The I²C protocol requires two wires (SCK and SDA). The pins PD[1:0] are configured as the I²C bus when the MDI2C bit is set in the pin configuration register, PCONF. The following table describes how each pin is connected:

Table 14 I²C Interface Signals

Name	Pin Number	Pin Name	Comments
SDA	37	PD1	Data
SCL	38	PD0	Clock

11.5.1 I2C Functionality

The I²C interface may be configured as a slave or master. The configuration procedures are described in the following sections.

11.5.1.1 Slave Mode

In order to configure Slave Mode, the following steps must be carried out

- Step 1: Select the peripheral as slave. Code Example:

```
I2C_Mode( I2C_SLAVE );
```

- Step 2: Select the address size. Code Example selecting 7-bit address size:

```
I2C_Slave_Sets_Address_Size ( 7_BIT_ADDRESS );
```

- Step 3: Load the address. Code example loading 0x7A as address:

```
I2C_WriteAddress(0x7A);
```

- Step 4: Select if general call is to be supported. Code example disabling general call:

```
I2C_Slave_Enable_General_Call ( GC_OFF );
```

- Step 5: Select address masking if required. If required the peripheral provides a 5-bit address mask for the lower 5 address bits. Each bit masks the corresponding bit in address comparison when set. For example, configure as an I²C Slave in 7-bit address mode is using 0x03 as mask and 0x36 as address Then the I²C will answer to all messages with addresses 0x34, 0x35, 0x36 or 0x37. Code example selecting 0x03 as mask:

```
I2C_Slave_Address_Mask ( 0x03 );
```

- Step 6: Enable the interface and its interrupts if used. Code example:

```
I2C_Enable( I2C_ON );//Enabling the interface
NVIC_EnableIRQ( I2C_IRQn );          //Enable I2C communications interrupt
NVIC_EnableIRQ( I2C_Collision_IRQn ); //Enable I2C collision interrupt
```

- Step 7: Define the Interrupt handlers if required. Code example:

```
void I2C_Collision_Handler( void )//
{
//Collision handling code
}

void I2C_Handler( void )//
{
//Communications code
}
```

In the following paragraphs the several phases of the slave side of the communications will be described.

11.5.1.1.1 I²C Access Sequence, Slave Mode

Note: I²C address phases are always prefixed by Start or Repeated Start condition.

11.5.1.1.2 7-bit Address mode

The slave, once enabled, waits for an I²C Start condition to happen. Once a Start condition is detected, the slave shifts in the next 8 bits into an internal shift register and the following actions take place:

- The **IBUFF** bit is set.
- If the address contained in the internal register matches the one from the register, **I2CADDR0** the interface sends back an ACK and an interrupt is asserted.
- At this moment the application must read the **I2CSTATUS** register and check the **IADDRR** and **IRWBUSY** bits.
- The **IADDRR** should be '1' (indicating that an address has been received).
- The **IRWBUSY** bit indicates if the operation is a write ('1') or a read ('0').
- After reading these bits the application must read the **I2CDATA** register to clear the buffer.
- If **IBUFF** is set before receiving the address or **IRBUFOVL** is set when receiving the address, then the slave will send NACK and issue an error interrupt to notify the application about these errors.

11.5.1.1.3 10-bit Address mode

Two address-byte receptions are required in this mode. The first byte consists of the following sequence

'1 1 1 1 0 A[9] A[8] 0' where A[9:8] are the upper two bits of I²C address.

The last bit must be 0 (R1W0) so the slave can receive another address byte. If the upper two address bits match then the Slave sends an ACK and asserts an interrupt.

The application must at this point read the **I2CSTATUS** register to check the **IADDRR** and **IRWBUSY**, which should be 1 (address byte) and 0 (write operation). The user then needs to read the **I2CDATA** register to clear the buffer.

The second byte contains the address bits A[7:0]. In the same fashion if the lower 8 address bits match then the slave sends the ACK and asserts an interrupt.

The application reads the **I2CSTATUS** register to check the **IADDRR** bit, which should be a logic 1 (address byte). The user then needs to read **I2CDATA** register to clear the buffer.

If it is an I²C read access, then after the two address-byte receptions the slave shall receive a Repeated Start condition and then the first address byte again with last bit (R1W0) being 1. The slave sends the ACK bit and asserts an interrupt
The application reads **I2CSTATUS** register to check the **IBUFF** and **IRWBUSY** bits, which should be 1 (address byte) and 1 (read operation). The user then needs to read the **I2CDATA** register to clear the buffer.

Figure 7 shows an example of 10-bit address mode receiving timing waveform.

11.5.1.1.4 I²C Access Sequence – Write Data Phase

If the received R1W0 field is 0, it is an I²C write access and the slave remains in receiving mode. Every time the slave shifts in a byte, it sends the ACK bit as long as the **IBUFF** bit is cleared before receiving the data and the **IRBUFOVL** bit is cleared when receiving the data.

In slave mode, the peripheral asserts an interrupt after receiving each byte from I²C bus. The user needs to read the **I2CSTATUS** register to check the status and then the **I2CDATA** register to fetch the data. The write data phase is concluded when detecting a Stop or Repeated Start condition.

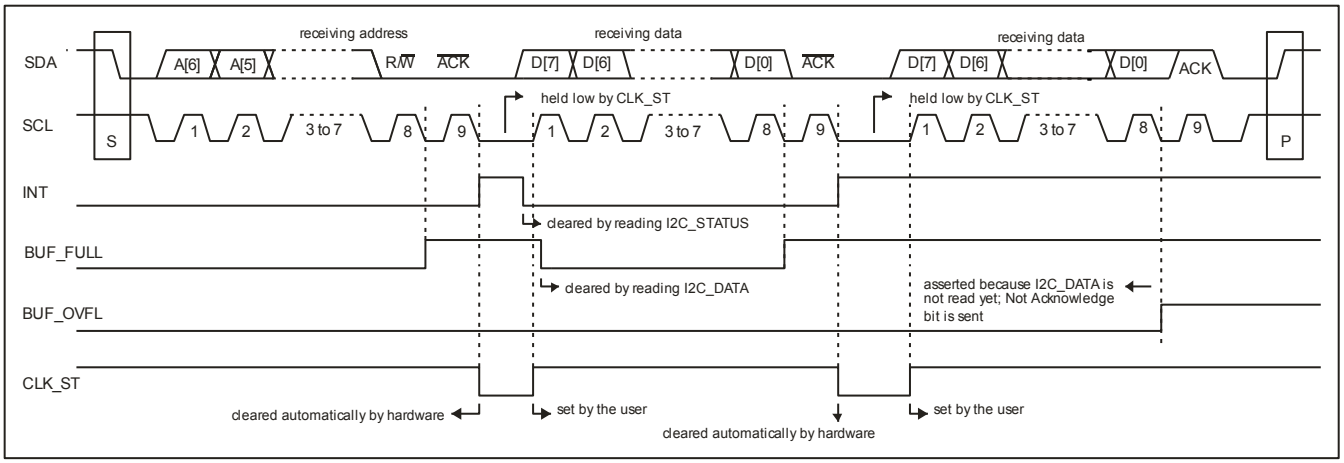


Figure 4 Slave Mode Timing Waveform with CLK_ST_ENB = 1 (Reception, 7-bit Address Mode)

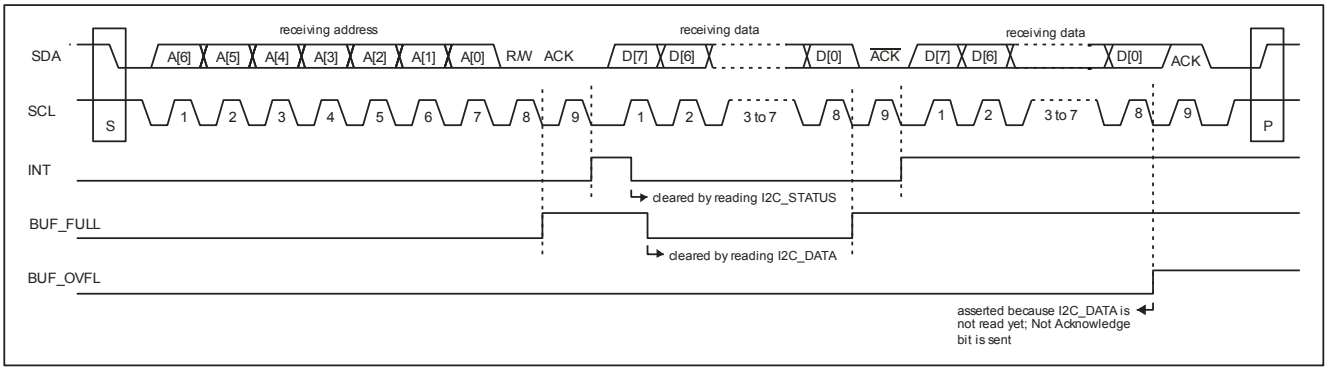


Figure 5 Slave Mode Timing Waveform with CLK_ST_ENB = 0 (Reception, 7-bit Address Mode)

11.5.1.1.5 I²C Access Sequence – Read Data Phase

If the received R1W0 field is 1, it is an I²C read access and the slave switches to transmitting mode.

Before each byte shifts out, the **ICLKSTR** bit is cleared automatically to hold the SCL pin low (known as clock stretching). The user needs to program the **I2CDATA** register with the byte to be transmitted and then set **ICLKSTR** bit to release SCL pin. Every time the slave shifts out a byte, it receives the ACK/NACK bit. If it receives the ACK bit, the slave clears the **ICLKSTR** bit automatically, and the user needs to program the **I2CDATA** register and set the **ICLKSTR** bit to resume transmission. If it receives the NACK bit, which means the Master device has completed reading data, the Slave releases both SCL and SDA. The Slave asserts an interrupt after receiving the ACK/NACK bit. The read data phase is concluded when receiving the NACK bit or detecting Repeated Start or Stop condition. Figure 6 and show examples of I²C read accesses.

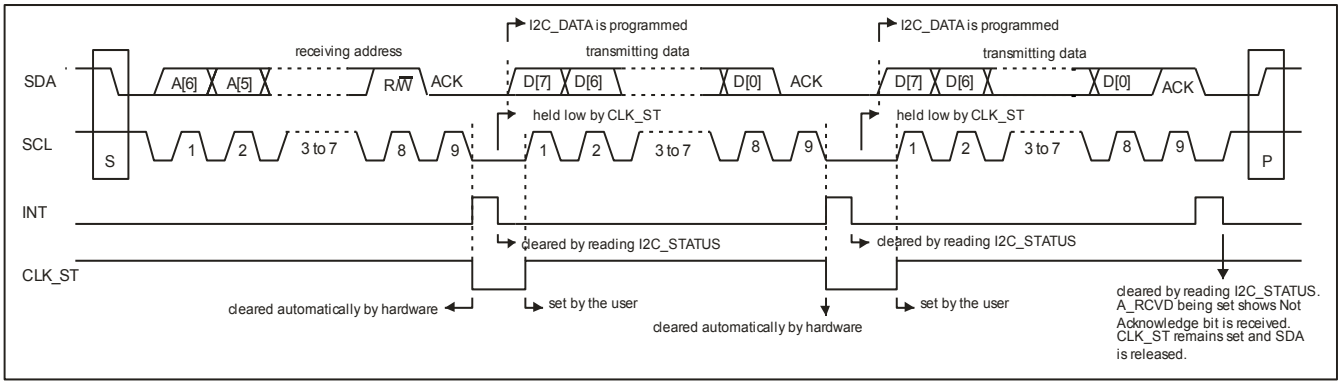


Figure 6 Slave Mode Timing Waveform (Transmission, 7-bit Address Mode)

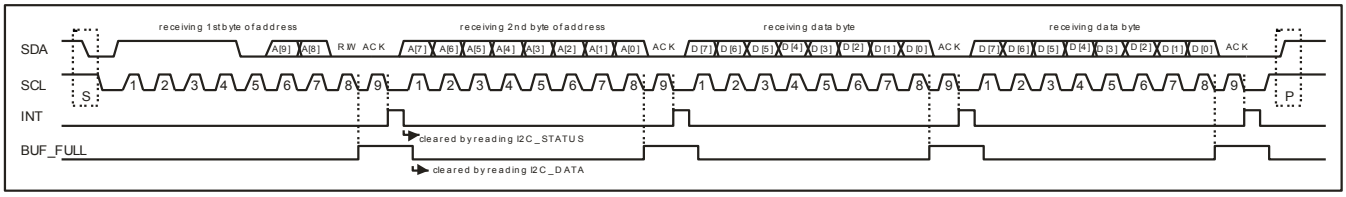


Figure 7 Slave Mode Timing Waveform (Reception, 10-bit Address Mode)

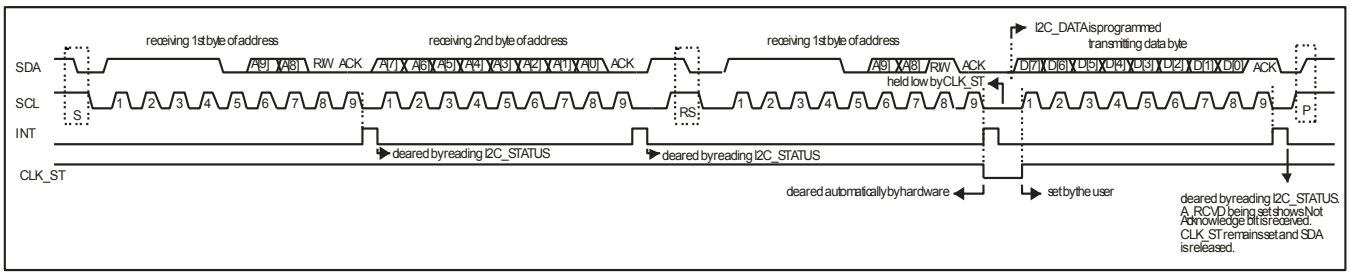


Figure 8 Slave Mode Timing Waveform (Transmission, 10-bit Address Mode)

11.5.1.2 Master Mode

In order to configure Master Mode, the following steps must be carried out:

- Step 1: Select the master mode. Code example:

I2C_Mode(I2C_MASTER);

- Step 2: Define the baud rate. The equation defining the baud rate as a function of the system clock is:

$$Fi2c = \frac{Fclk}{2 * (divider + 1)}$$

Where Fi2c is the frequency of the I^2C interface, divider is the i2c divider and Fclk is the system clock. Code example to select an I^2C clock of 100kbits/sec. (Assuming a system clock of 4MHz)

I2C_WriteClockDivider(19); //I2C baud = 100kbits/sec

- Step 3: Enable the interface. Code example:

I2C_Enable(I2C_ON);//Enabling the interface

- Step 4: Enable the interrupts and define the corresponding handlers, exactly like steps 6 and 7 from the slave mode.

From this point on the application software must handle the communication. The following paragraphs will describe the general steps:

11.5.1.2.1 Configuration Settings

As an I^2C Master, the Master controls SCL and SDA when issuing Start, Repeated Start and Stop conditions. It also drives (release/drain) SCL and SDA when transmitting address/data bytes as well as ACK/NACK bits after receiving data bytes.

When the **IEN** and **IMS** bits are both set in the **I2CCTRL2** register the interface is configured as an I^2C Master.

11.5.1.2.2 Baud Rate Generators Configuration

A baud rate generator (BRG) inside the peripheral serves as an engine to time SCL transitions during data transfer as well as the transitions of both SCL/SDA during Start, Repeated Start and Stop conditions.
The BRG consists of an 8-bit counter that when enabled, loads the value from the **I2CADDR0** register and counts down to 0. Then it goes back to **I2CADDR0** register value and repeats the counting down process. When the BRG counter counts down to 0, it triggers the SCL transitions during data transfer and SCL/SDA transitions during Start, Repeated Start and Stop conditions.
The peripheral's baud rate is determined by the system clock frequency F_{clk} and divider.
The equation is:

$$Fi2c = \frac{Fclk}{2 * (divider + 1)}$$

11.5.1.2.3 Start Condition

The Master issues a Start condition when the **IRSTRB** bit is set by the user. When detecting the issued Start condition the Master asserts an interrupt and clears the **ISTRSTRETCH** bit. The user reads the **I2CSTATUS** register to clear the interrupt condition. At this point the **ISTRR** bit is set.

Once the **ISTRSTRETCH** bit is set, if SCL is sampled low before SDA goes low or if SCL or SDA is already sampled low when the **ISTRSTRETCH** bit is set, then it is concluded that there is a bus collision due to another I^2C Master on the bus, and the bus collision interrupt is asserted. The application must read the **I2CSTATUS** register to clear this interruption condition.

11.5.1.2.4 Repeated Start Condition

The Master issues a Repeated Start condition when the **ISTRSTRETCH** bit is set by the user. When detecting the issued Repeated Start condition, the Master asserts an interrupt and clears the **IRSTR** bit. The user reads the **I2CSTATUS** register to clear the interruption condition. At this point the **ISTRR** bit is set.

Once the **IRSTR** bit is set, if SCL is sampled low first before SDA goes low, or if SDA is sampled low when SCL goes from low to high, then it is concluded that there is an I^2C bus collision and a collision interrupt is asserted. The user reads **I2CSTATUS** register to clear the interrupt condition.

11.5.1.2.5 Stop Condition

The Master issues a Stop condition when the **ISTPSIZE** bit is set by the user. When detecting the issued Stop condition, the Master asserts an interrupt and clears the **ISTPSIZE** bit. The user reads the **I2CSTATUS** register to clear the interrupt condition. The **ISTPR** bit is set.

Once the **ISTPSIZE** bit is set, if SDA is sampled low one baud period (T_{br}) after it is released by the peripheral, or after SCL is released, SCL is sampled low before SDA goes high, then it is concluded that there is an I^2C bus collision and a collision interrupt is asserted. The user reads the **I2CSTATUS** register to clear the collision interrupt condition.

11.5.1.2.6 Acknowledge Bit

The Master transmits ACK/NACK bit when **ISACK** is set by the user. If the value of **ISNACK** is 1, a NACK bit is transmitted otherwise an ACK bit is transmitted. The Master asserts an interrupt and clears **ISNACK**. The user reads **I2CSTATUS** to clear the interrupt condition.

If the Master transmits a NACK bit but detects an ACK bit, Then There is an I^2C bus collision and a collision interrupt is asserted. The user reads **I2CSTATUS** to clear the collision interrupt.

11.5.1.2.7 I2C Write Access Sequence

The typical I²C write access sequence consists of the following steps:

- Step 1: The user sets the **ISTRSTRETCH** bit to issue a Start condition.
- Step 2: The Master detects the Start condition and asserts an interrupt. The user reads the **I2CSTATUS** register to clear the interrupt and check the **ISTRR** bit.
- Step 3: The user programs the **I2CDATA** register with the appropriate I²C Slave's address, then the Master starts transmitting the address byte.
- Step 4: After sampling the ACK/NACK bit sent by the Slave, the Master asserts an interrupt. The user reads the **I2CSTATUS** register to clear the interrupt condition and check **IACKR** bit.
- Step 5: The user programs the **I2CDATA** register with the data byte to be transmitted, then the Master starts transmitting the data byte.
- Step 6: After sampling the ACK/NACK bit sent by the Slave, the Master asserts an interrupt. The user reads **I2CSTATUS** register to clear the interrupt condition and check **IACKR** bit.
- Step 7: Repeat steps 5 and 6 to transmit more bytes.
- Step 8: The user can access a different I²C Slave or read from the same one by setting the **IRSTR** bit. This causes the Master to issue a Repeated Start condition. When the condition is sampled the Master asserts an interrupt. The user reads the **I2CSTATUS** register to clear the interrupt condition and check the **ISTRR** bit.
- Step 9: The user concludes the current transfer by setting the **ISTPSIZE** bit, and the Master issues a Stop condition. When the condition is sampled, the Master asserts an interrupt. The user reads the **I2CSTATUS** register to clear the interrupt and check the **ISTPR** bit.

Figure below shows a timing waveform of Master write access. Note that the only difference between 7-bit and 10-bit address mode is that the user needs to program two address bytes in 10-bit mode.

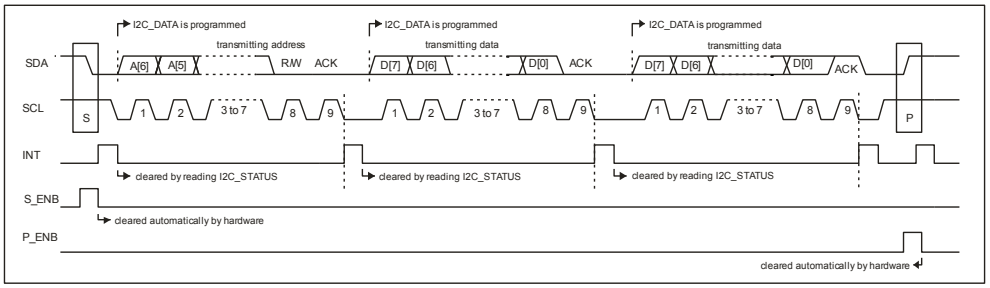


Figure 9 Master Timing Waveform (transmission)

11.5.1.2.8 I²C Read Access Sequence

The typical I²C read access sequence consists of the following steps:

- Step 1: The user sets the **ISTRSTRETCH** bit to issue a Start condition.
- Step 2: The Master detects the Start condition and asserts an interrupt. The user reads the **I2CSTATUS** register to clear the interrupt and check the **ISTRR** bit.
- Step 3: The user programs the **I2CDATA** register with the appropriate I²C Slave's address, then the Master starts transmitting the address byte.
- Step 4: After sampling the ACK/NACK bit sent by the Slave, the Master asserts an interrupt. The user reads the **I2CSTATUS** register to clear the interrupt and check the **IACKR** bit.
- Step 5: The user sets the **IRCSTRT** bit, which enables the Master to pulse the SCL pin and shift in a data byte. After shifting in the whole data byte, the Master asserts an interrupt. The user reads the **I2CSTATUS** register to clear the interrupt. The user then reads the **I2CDATA** register to fetch the received data byte.
- Step 6: The user clears the **ISNACK** bit (Acknowledge bit to be sent) and sets the **ISACK** bit. The Master transmits the ACK bit. After the transmission, the Master asserts an interrupt. The user reads the **I2CSTATUS** register to clear the interrupt.
- Step 7: Repeat steps 5 and 6 to receive more bytes.
- Step 8: The user can access a different I²C Slave or write to the same one by setting the **IRSTR** bit, then the Master issues a Repeated Start condition. When the condition is sampled the Master asserts an interrupt. The user reads the **I2CSTATUS** register to clear the interrupt and check the **ISTRR** bit.
- Step 9: If the data byte being received is the last one, after clearing the interrupt, the user sets the **ISNACK** (Not Acknowledge bit to be sent) and **IRSTS** bits. The Master transmits NACK bit. After the transmission the Master asserts an interrupt. The user reads the **I2CSTATUS** register to clear the interrupt.
- Step 10: The user concludes current transfer by setting the **ISTPSIZE** bit, and the Master issues a Stop condition. When the condition is sampled the Master asserts an interrupt. The user reads the **I2CSTATUS** register to clear the interrupt and check the **ISTPR** bit.

Figure 10 shows an example of Master read access.

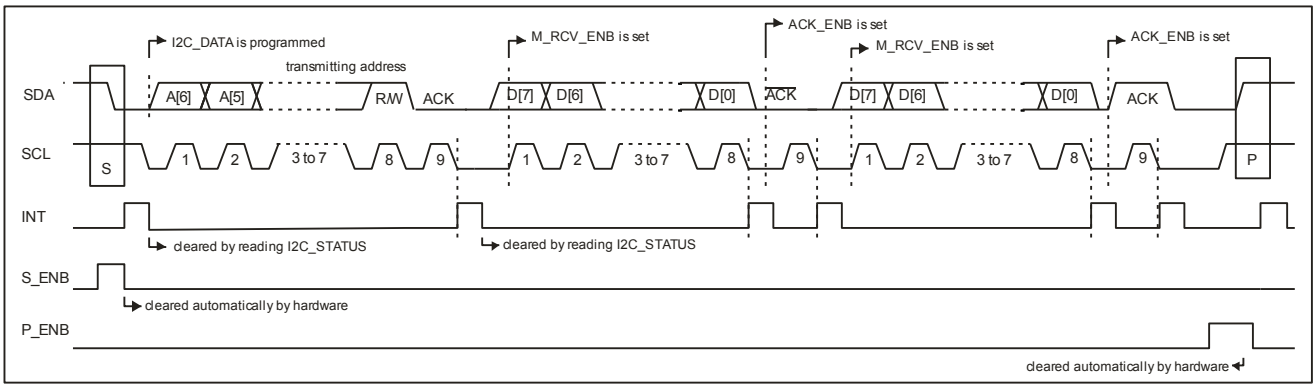


Figure 10 Master Timing Waveform (Reception)

11.5.1.2.9 RW_BUSY Indicator

Whenever detecting a Start/Stop condition, which it did not issue itself, the Master asserts/de-asserts RW_BUSY to indicate the busy/idle status of the I^{2}C bus. The user can check the **IRWBUSY** bit before issuing a Start condition so bus collision can be avoided.

11.5.2 I2C Registers

The following registers are used to control the I^{2}C interface.

Table 15 I2C Control Register Map

Address	Register Name	Description	Reset Value	Reference
0x50000008	I2CSTATUS	I2C Status Register	0x00	Register 19
0x50000009	I2CCTRL1	I2C Control Register 1	0x40	Register 20
0x5000000A	I2CCTRL2	I2C Control Register 2	0x02	Register 21
0x5000000B	I2CDATA	I2C Data Register	0x00	Register 22
0x5000000C	I2CADDR0	I2C Address Register 0	0x00	Register 23
0x5000000D	I2CADDR1	I2C Address Register 1	0x00	Register 24

Register 19 I2C Status Register							
I2CSTATUS		0x50000008			0x00		
R	R	R	R	R	R	R	R
IACKR	IADDRR	ISTRR	ISTPR	IRWBUSY	IBUFF	IWBUFOVL	IRBUFOVL
MSB							LSB

Bit7 **IACKR**: Acknowledge received

0 = ACK received

1 = ACK not received

Bit6 **IADDRR**: Data/Address received (slave mode)

0 = DATA received

1 = ADDRESS received

Bit5 **ISTRR**: Start bit received

0 = Start bit not received

1 = Start bit received

Bit4 **ISTPR**: Stop bit received (slave mode)

0 = No stop bit received

1 = Stop bit received

Bit3 **IRWBUSY**: Read/Write Busy:

Master Mode:

0 = Bus not being accessed

1 = Bus being accessed

Slave Mode:

0 = I2C write operation (Slave receives data)

1 = I2C read operation (Slave transmits data)

Bit2 **IBUFF**: Buffer Full

0 = Buffer is empty

1 = Buffer full, either because it received data or there is one byte to be transmitted

Bit1 **IWBUFOVL**: Write buffer overflow

Register 19 I2C Status Register
0 = Buffer is empty 1 = Internal shift register is full and the **I2CDATA** register was written Bit0 **IRBUFOVL**: Data/Address received (slave mode) 0 = Register was read 1 = Internal shift register is full and another byte is received from I2C bus. NOTE: While **IRBUFOVL** is set the shift-in of bits from bus is stopped.

Register 20 I2C Control Register 1							
I2CCTRL1		0x50000009			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
IRSTR	ICLKSTR	IGC	IRCSTRT	ISNACK	ISACK	ISTPSIZE	ISTRSTRETCH
MSB							LSB

Bit7 **IRSTR**: Repeated start bit (Master Only)

0 = Repeated start bit disabled

1 = Issue start-bit transmit enable (when set the I2C transmits Repeated Start bit), cleared by HW.

Bit6 **ICLKSTR**: Clock stretch (Slave Only)

0 = SCL held low - 1 = SCL released

Bit5 **IGC**: General call address (Slave Only)

0 = General call address disabled

1 = General call address enabled

Bit4 **IRCSTRT**: Start bit reception (Master Only and cleared by HW)

0 = Receive operation not allowed

1 = Receive operation starts (the receive operation starts when this bit is set)

Bit3 **ISNACK**: ACK bit to be transmitted: (Master Only)

0 = ACK is transmitted upon reception of byte

1 = NACK is transmitted upon reception of byte

Bit2 **ISACK**: ACK bit (Master Only)

0 = No ACK/NACK bit transmitted

1 = Acknowledge (ACK/NACK, defined by ISNACK) is transmitted

Bit1 **ISTPSIZE**: Stop bit or selection of address size

Master Mode

0 = No stop bit sent - 1 = Stop bit sent

Slave Mode

0 = 7-bit address - 1 = 10-bit address

Bit0 **ISTRSTRETCH**: Start and stretch

Master Mode

0 = No start bit sent - 1 = Send start bit

Slave Mode

0 = no clock stretch - 1 = Clock stretched

Register 21 I2C Control Register 2

I2CCTRL2		0x5000000A			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
IMSK4	IMSK3	IMSK2	IMSK1	IMSK0	IEN	IFILTER	IMS
MSB							LSB

Bit7-3 **IMSK[4:0]**: I^2C Address mask (Slave Only)

Bit2 **IEN**: I^2C Enable bit

0 = I^2C disabled

1 = I^2C enabled

Bit1 **IFILTER**: I^2C filter

0 = Filter disabled

1 = Filter enabled

Bit0 **IMS**: I^2C Master/Slave

0 = I^2C slave

1 = I^2C master

NOTE: In order to ignore the glitches on I^2C bus, a 3-tab median filter operating at system clock rate is implemented on the inc

SCL and SDA data paths. This filter can be enabled/disabled by setting/clearing IFILTER. The truth table of the median filter is below.

Filter Tabs and Output			
Filter tab 0	Filter tab 1	Filter tab 2	Filter output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Register 22 I2C Data Register							
I2CDATA		0x5000000B			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
IDT7	IDT6	IDT5	IDT4	IDT3	IDT2	IDT1	IDT0
MSB							LSB
Bit7-0 **IDT[7:0]**: I^2C Data							

Register 23 I2C Address 0 Register							
I2CADDR0		0x5000000C			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
IADDR7	IADDR6	IADDR5	IADDR4	IADDR3	IADDR2	IADDR1	IADDR0
MSB							LSB
Bit7-0 **IADDR[7:0]**: In Master mode, the data represents the 8-bit clock frequency divider counter maximum value. In Slave mode, it represents the lower 8-bit I2C address.							

Register 24 I2C Address 1 Register							
I2CADDR1		0x5000000D			0x00		
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	R/W	R/W
-	-	-	-	-	-	IADDR9	IADDR8
MSB							LSB
Bit7-0 **IADDR[9:8]**: In Slave mode, the data represents the upper 2-bit I2C address. In Master mode, the data is ignored.							

11.6 ADC

Kamcho includes an analog to digital converter (ADC). The ADC is an 8-bit analog to digital converter with single ended input. The main features are described below:

- 8-bit resolution
- Single ended input
- Up to 80 ksps
- Configurable reference (VREF = VREFHI-VREFLO)
    - Either based on the bandgap voltage(VBG) or supply voltage(VDD)
    - Reference may be scaled in order to provide more resolution around a smaller input voltage range
- Maximum ADC input range is from 0V to its supply voltage
- It may read from a total of 40 channels (39 GPIOs and a PTAT reference in order that temperature may be measured)
- Using VBG as the reference, supply voltage can be measured in calibration mode

11.6.1 ADC Description

Kamcho ADC uses a standard charge redistribution technique, with a single-ended input and internally generated positive and negative reference voltages. The ADC accommodates 40 analog input channels. The user can select which input channel to be sampled by setting the ADCCHANNEL register. The ADC has its own internally generated reference voltages (VREFHI and VREFLO). The performance table is shown below.

Table 16 ADC Performance Specification, Recommended Operating Conditions, unless otherwise specified

Parameter	Conditions	min	typ	max	unit
Conversion speed	Signal source resistance less than 20kΩ			80	ksps
Clock Frequency				1	MHz
Input voltage range		0		VDD	V
Resolution				8	bits
INL				1	LSB
DNL				1	LSB

There are several steps required for the user to use the ADC. The general sequence is described below:

- Step 1: Select the input channel to be measured
- Step 2: Configure ADC settings.
    a. Set ADC clock frequency.
    b. Configure references by programming the ADCREFHI, ADCREFLO, ADCPGN, and ADCREFS bits.
- Step 3: Start the ADC conversion.
- Step 4: Check the ADC status bit and read the data.

The following section will describe each configuration step in detail.

11.6.1.1 Input Channel Selection

All GPIOs (PA[7:0], PB[7:0], PC[7], PC[5:0], PD[7:0] and PE[7:0]) and the PTAT reference are available as an inputs to the ADC. The user can control which input is connected for the conversion by programming the control bits, **ADDCH[5:0]**, in the **ADDCHANNEL** register.

Code Example: Selecting PA0 for ADC input channel

```
ADC_Select_Channel( ADC_PA0 );
```

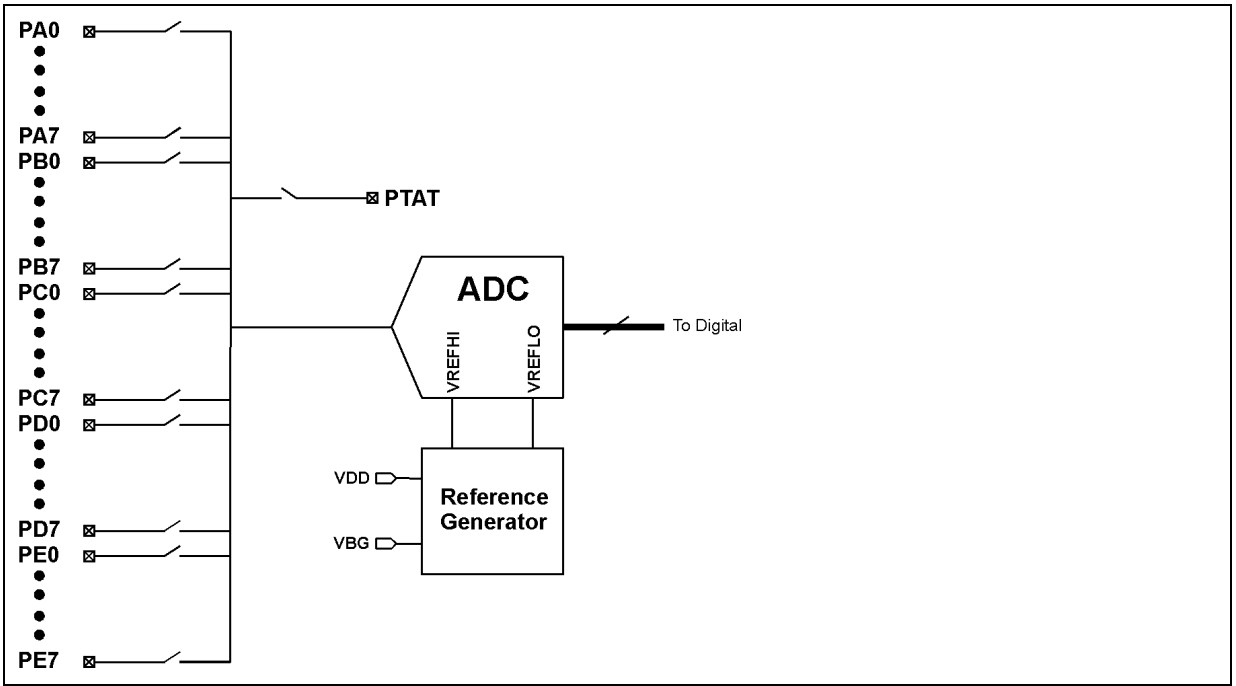


Figure 11 ADC Block Diagram

11.6.1.2 ADC Clock and Sampling Period

The conversion algorithm has a basic period of 9 cycles (one for sampling, one for each bit). There is a two-cycle latency from the last bit measurement until the data becomes available to the user. Additionally, there is a single idle cycle to allow biasing before any conversion is initiated. Thus a single conversion will take 13 cycles.

The converter will use a single clock cycle to sample the input into an input capacitor. When the channel is selected, the source must drive the sample/hold capacitor through the source resistance of the signal to be measured. The sampling time varies with this source resistance. The input to ADC must have sufficiently low driving impedance and settling time to settle the input to within 1 LSB of the data conversion during the input sampling stage. An equivalent circuit and related equations are depicted in Figure 12.

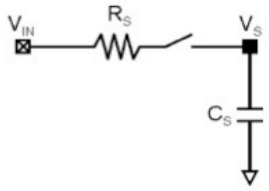
The ADC clock frequency can be programmed through the **ADCCLKDIV** register. As an example, if ADC clock is derived from a 12MHz oscillator divided by 16, the input has 1.3µs to settle. Since the maximum value of the internal sample/hold capacitor, C_s is 10pF, the maximum source resistance of the signal to be sampled to guarantee 8-bit performance can be calculated as below:

$$R_s = \frac{1.3\mu s}{10pF \times 5.5} = 24.2k\Omega$$

If the source impedance is larger, the user can reduce the ADC clock frequency in order to retain the conversion accuracy.

Code Example: Configure ADC clock to 12MHz RC oscillator frequency divided by 16.

```
ADC_ClkDiv (16);
```



$$V_S = V_{IN}(1 - e^{-\frac{t}{\tau}}) \quad , where\ \tau = R_S C_S$$

V_S *is within 1 LSB of* V_{IN} *after* $t = 5.5\tau$

Figure 12 ADC Input Settling Time

11.6.1.3 Configuration of Reference Voltages for the ADC

The ADC can generate its own reference voltages (VREFHI and VREFLO) from two different sources, its supply voltage (VDD) or the internal bandgap reference voltage (VBG). The **ADCREFS** bit in the **ADCREG3** register selects the source. Once the reference source is selected, the reference voltages can be programmed through the **ADCREFHI**, **ADCREFLO**, and **ADCPGN** bits according to Figure 13.

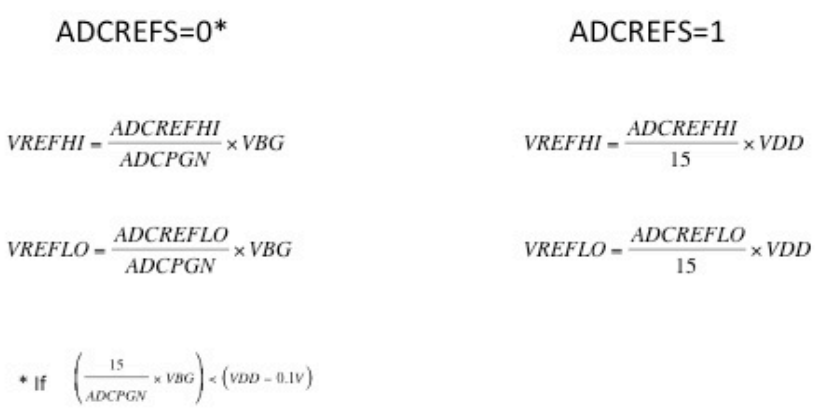


Figure 13 ADC Reference Voltage

Once the reference voltages are established, the ADC conversion equation for input voltage (VIN) can be defined as:

$$ADCDT = floor\left(255 \times \frac{(VIN - VREFLO)}{(VREFHI - VREFLO)}\right)$$

Here are a few examples:

- Example 1: In a system operating with VDD=3V, there is a signal that moves between 0V and 2.94V. In this case it is recommended that VDD be selected as the reference source and the following setting be made, **ADCREFH**=15, **ADCREFL**=0, and **ADCPGN**=15. This selection would allow for the maximum range of measurement (0V to VDD).
    - Code Sample ADC_Reference_Config(ADC1, 15, 0, 15, ADCREFVDD);
- Example 2: In a system operating with VDD=3V and VBG= 1.21V, there is a signal that moves between 0V and 2.5V. In this case VBG can be selected as the reference source with settings of **ADCREFH**=13, **ADCREFL**=0 and **ADCPGN**=7. This configuration would allow the signal range of measurement to be 0V to 2.6V:
    - Code Sample ADC_Reference_Config(13, 0, 7, ADCREFBG);
- Example 3: In a system operating with VDD=3V and VBG=1.21V, there is a signal that moves between 1.71V and 2.2V. In this case selecting VBG as the reference source and settings of **ADCREFH**=15, **ADCREFL**=11, and **ADCPGN**=8 we can achieve higher resolution:

- Code Sample ADC_Reference_Config(15, 11, 8, ADCREFBG);

The resolution in Example 3 can be calculated as follows:

$$RESOLUTION = \left(\frac{VREFHI - VREFLO}{255}\right) = \left(\frac{\left(\frac{15}{8}\times 1.21\right) - \left(\frac{11}{8}\times 1.21\right)}{255}\right) = \frac{2.27 - 1.66}{255} = 2.38mV$$

Note that in this particular case we have the 8-bit ADC effectively generating a digital value with the precision of a 10-bit ADC operating from 0V to VDD.

It is clear from the examples how flexible the ADC can be in a range of applications. The user can devise several schemes to cleverly measure the range of signal of interest and then narrow the reference values to get the optimum resolution if the conversion time is acceptable.

11.6.1.4 ADC Start and Status

Before starting the conversion, the ADC must be enabled and biased. The ADC is enabled by the **ADCEN** bit in register **ADCREG3**. The START bit (**ADCSTART**) starts the conversion process. Once completed the value of the conversion is loaded into the **ADCDATA** register.

Code Example: Enable the converter and start conversion

```
ADC_Enable(ADCEN);
ADC_Start();                              //ADC enabled and start
while ( ADC_ConversionComplete() == 1 );  //Wait until completed
```

11.6.2 ADC Registers

The following registers control the behavior of the ADC:

Table 17 ADC Control Register Map				
Address	Register Name	Description	Reset Value	Reference
0x50000058	ADCCHANNELS			
0x50000059	ADCSTART			
0x5000005A	ADCDATA			
0x5000005B	ADCCLKDIV			
0x50018010	ADCTRIM0		0xF0	
0x50018011	ADCTRIM1		0x9F	
0x50018012	ADCTRIM2		0x00	

Register 25 ADC Channel Select Register							
ADCCHANNELS		0x50000058			0x00		
Reserved	Reserved	R/W	R/W	R/W	R/W	R/W	R/W
-	-	ADCCH5	ADCCH4	ADCCH3	ADCCH2	ADCCH1	ADCCH0
MSB							LSB

Bit5-0 **ADCCH[5:0]:** ADC Channel Select

000000 - 000111 = PA0 through PA7

001000 - 001111 = PB0 through PB7

010000 - 010111 = PC0 through PC7 (no PC6)

011000 - 011111 = PD0 through PD7

100000 - 100111 = PE0 through PE7

101111 = PTAT

Register 26 ADC Start Register							
ADCSTART		0x50000059			0x00		
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	R/W
_	_	_	_	_	_	_	START
MSB							LSB
Bit0 **START**: Writing one starts the conversion. Reading returns the status of conversion; '0' means conversion Is finished and '1' means the conversion is ether pending or in progress							

Register 27 ADC Result Register							
ADCDATA		0x5000005A			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADCDT7	ADCDT6	ADCDT5	ADCDT4	ADCDT3	ADCDT2	ADCDT1	ADCDT0
MSB							LSB
Bit7-0 **ADCDT[7:0]**: ADC Result							

Register 28 ADC Clock Divider Control Register							
ADCCLKDIV		0x5000005B			0x6F		
Reserved	R/W	Reserved	Reserved	R/W	R/W	R/W	R/W
-	-	-	-	-	ADCDIV1	ADCDIV0	ADCCONT
MSB							LSB
Bit2-1 **ADCDIV[1:0]:** ADC Clock divider 00 = System Clock/8 01 = System Clock/16 10 = System Clock/32 11 = System Clock/64 Bit0 **ADCCONT:** ADC Stream Mode 0 = Single Conversion - 1 = Streaming Mode							

Register 29 ADC Trim0 Register							
ADCTRIM0		0x50018010			0xF0		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADCREFH3	ADCREFH2	ADCREFH1	ADCREFH0	ADCREFL3	ADCREFL 2	ADCREFL 1	ADCREFL 0
MSB							LSB

Bit7-4 **ADCREFH[3:0]:** ADC Reference High Setting

Bit3-0 **ADCREFL[3:0]:** ADC Reference Low Setting

Register 30 ADC Trim1 Register							
ADCTRIM1		0x50018011			0x9F		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ADCSW1	ADCSW0	ADCCAL	ADCREFS	ADCPGN3	ADCPGN2	ADCPGN1	ADCPGN0
MSB							LSB

Bit7-6 **ADCSW[1:0]:** ADCs Enable Bit.

00 = Correlated double sampling off

01 = Input offset calibration on

1x = Correlated doubling sampling on (default)

Bit5 **ADCCAL:** ADC Calibration

0 = Normal

1 = Calibration mode

Bit4 **ADCREFS:** ADC Internal Reference Source Selection

0 = Band Gap

1 = VDD

Bit3-0 **ADCPGN[3:0]:** ADC Reference Gain

Register 31 ADC Trim2 Register							
ADCTRIM2		0x50018012			0x00		
Reserved	Reserved	Reserved	R/W	R/W	R/W	R/W	R/W
-	-	-	ADCGNDOFF	ADCENFORC	ADCSCYC2	ADCSCYC1	ADCSCYC0
MSB							LSB

Bit4 **ADCGNDOFF:** Ground offset correction

0 = bandgap and ADC reference have common ground

1 = difference between bandgap and ADC reference is compensated by switched capacitor circuit

Bit3 **ADCENFORC:** Manual Sampling

0 = Normal

1 = Calibration mode

Bit2-0 **ADCSCYC[2:0]:** ADC Sampling cycle (Sampling period = ADCSCYC + 1 cycles)

11.7 Pulse Width Modulators (PWM)

Kamcho includes two PWMs. Their main characteristics are:

- Twelve bit resolution – Both period and pulse width.
- Independent Prescalers
- Programmable active level
- Programmable outputs
    - PWM1 may be connected to any pin on port A, (pins PA[7:0])
    - PWM2 may be connected to any pin on port C, (pins PC[7], PC[5:0])

11.7.1 PWM Usage Description

The PWM circuit generates a wide-range, high-resolution modulated output. Each PWM has total of 4 data and configuration registers to communicate with the microcontroller.

The waveform is controlled by 12-bit period word (PWMnPER and PWMnEXT) and 12-bit pulse width word (PWMnPW and PWMnEXT) are used to determine the output waveform.

The entire waveform can be scaled by adjusting the Prescaler value in PWMnCTRL. The Prescaler can be set to one of eight different settings shown in Table 18.

Table 18 PWM Prescaler Divider Values	
PRESC[2:0]	**Divide Value (f_{XO}/f_{PWM})**
000	1
001	2
010	4
011	8
100	32
101	256
110	8,192
111	262,144

The output period is calculated as follows:

$$Period = \frac{1 + (PWMPER \times DIVIDE\_VALUE)}{SystemClock}$$

The PWM pulse width is calculated as follows:

$$Pulse\_Wdith = \frac{1 + (PWMPW \times DIVIDE\_VALUE)}{SystemClock}$$

To control the active level of the PWM, a control bit **PWM_INV** is used. If this bit is set to one, the PWM output is low level during the pulse and one at other times, including if the PWM is disabled by the user.

Alternatively if **PWM_INV** is set to zero then the PWM outputs a high level during the pulse.

Code Example: Setting the PWM1, enabled, with period and width separated, not inverted, with a PRESC[2:0] = 100:

```
PWM_Setup( PWM1, PWMEN, PWMNINV, PRESC4);
```

Code Example: Selecting the period and pulse width of the same PWM1.

```
PWM_Period (PWM1,200);
PWM_PW (PWM1,100);
```

NOTE: From the above equations, we can see that for a system clock of 12MHz the period is 533.4µsec and the pulse width is 266.8µsec.

11.7.2 PWMs Registers

The following registers are provided to control the PWMs:

Table 19 Pulse Width Modulator Control Register Map

Address	Register Name	Description	Reset Value	Reference
0x50000048	PWM1_CTRL	PWM 1 control	0x00	Register 40
0x50000049	PWM1_PER	PWM 1 period	0x00	
0x5000004A	PWM1_PW	PWM 1 pulse width	0x00	
0x5000004B	PWM1_EXT	PWM 1 period and pulse width	0x00	
0x5000004C	PWM2_CTRL	PWM 2 control	0x00	
0x5000004D	PWM2_PER	PWM 2 period	0x00	
0x5000004E	PWM2_PW	PWM 2 pulse width	0x00	
0x5000004F	PWM2_EXT	PWM 2 period and pulse width	0x00	

Register 32 PWM1 Control Register							
PWM1CTRL		0x50000048			0x00		
R/W	Reserved	Reserved	R/W	Reserved	R/W	R/W	R/W
PWM_EN	-	-	PWM_INV	-	PRESC2	PRESC1	PRESC0
MSB							LSB

Bit7 **PWM_EN:** PWM1 enable bit.

0 = PWM Disabled

1 = PWM Enabled

Bit4 **PWM_INV:** PWM output signal direction

0 = normal logic

1 = inverted logic (active low)

Bit2-0 **PRESC[2:0]:** PWM's Prescaler

000 = System Clock/1

001 = System Clock/2

010 = System Clock/4

011 = System Clock/8

100 = System Clock/32

101 = System Clock/256

110 = System Clock/8192

111 = System Clock/262144 (2^{18})

Register 33 PWM1 Period Low Byte Register							
PWM1PER		0x50000049			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM1PER7	PWM1PER6	PWM1PER5	PWM1PER4	PWM1PER3	PWM1PER2	PWM1PER1	PWM1PER0
MSB							LSB

Bit7-0 **PWM1PER[7:0]:** PWM1 period low register

Register 34 PWM1 Pulse Width Low Byte Register							
PWM1PW		0x5000004A			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM1PW7	PWM1PW6	PWM1PW5	PWM1PW4	PWM1PW3	PWM1PW2	PWM1PW1	PWM1PW0
MSB							LSB
Bit7-0 **PWM1PW[7:0]:** PWM1 width low register							

Register 35 PWM1 extension with high nibble of period and pulse width							
PWM1EXT		0x5000004B			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM1PER11	PWM1PER10	PWM1PER9	PWM1PER8	PWM1PW11	PWM1PW10	PWM1PW9	PWM1PW8
MSB							LSB
Bit7-4 **PWM1PER[11:8]:** PWM1 period high nibble Bit3-0 **PWM1PW[11:8]:** PWM1 width high nibble							

Register 36 PWM2 Control Register							
PWM2CTRL		0x5000004C			0x00		
R/W	Reserved	Reserved	R/W	Reserved	R/W	R/W	R/W
PWM_EN	-	-	PWM2_INV	-	PRESC2	PRESC1	PRESC0
MSB							LSB

Bit7 **PWM_EN:** PWM1 enable bit.

0 = PWM Disabled

1 = PWM Enabled

Bit4 **PWM_INV:** PWM output signal direction

0 = normal logic

1 = inverted logic (active low)

Bit2-0 **PRESC[2:0]:** PWM's Prescaler

000 = System Clock/1

001 = System Clock/2

010 = System Clock/4

011 = System Clock/8

100 = System Clock/32

101 = System Clock/256

110 = System Clock/8192

111 = System Clock/262144 (2^{18})

Register 37 PWM2 Period Low Byte Register							
PWM2PER		0x5000004D			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM2PER7	PWM2PER6	PWM2PER5	PWM2PER4	PWM2PER3	PWM2PER2	PWM2PER1	PWM2PER0
MSB							LSB

Bit7-0 **PWM2PER[7:0]:** PWM2 period low register

Register 38 PWM2 Pulse Width Low Byte Register							
PWM2PW		0x5000004E			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM2PW7	PWM2PW6	PWM2PW5	PWM2PW4	PWM2PW3	PWM2PW2	PWM2PW1	PWM2PW0
MSB							LSB
Bit7-0 **PWM2PW[7:0]:** PWM2 width low register							

Register 39 PWM2 extension with high nibble of period and pulse width							
PWM2EXT		0x5000004F			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM2PER11	PWM2PER10	PWM2PER9	PWM2PER8	PWM2PW11	PWM2PW10	PWM2PW9	PWM2PW8
MSB							LSB
Bit7-4 **PWM2PER[11:8]:** PWM2 period high nibble Bit3-0 **PWM2PW[11:8]:** PWM2 width high nibble							

11.8 GPIO Pins

Kamcho provides 39 general-purpose I/O pins that share functions with different peripherals. The following table defines the main characteristics of the GPIO pins:

Table 20 GPIO Main Characteristics

Recommended Operating conditions unless otherwise specified

Name	Conditions	Min	Typ	Max	Name	Conditions	Min	Typ	Max	Unit
V_{IL}	VBAT mode			0.3* VBAT	V_{IL}	VSUP mode			0.3* VSUP	V
V_{IH}		0.7* VBAT			V_{IH}		0.7* VSUP			V
I_{OL}			10		I_{OL}			10		mA
I_{OH}			10		I_{OH}			10		mA
PullDown			30		PullDown			30		⌠A
PullUp			30		PullUp			30		⌠A

11.8.1 GPIO Description

GPIOs may be used as a digital output, digital input, or analog input to the ADC.

Several pins have additional functions. Some of them may be connected to serial interface hardware or to a PWM circuit.

Ports A through E, have a register, which contains the data output to or input through the pins.

Pin direction, read or write, is set by a register for each port.

Each port has a read enable and a write enable register, and there are two registers per port, which control the activity of a pull up or pull down function (excluding PD0 and PD1), to ensure the pin is at a known state in input mode, even if the driving source is floating.

Additionally, each pin of port A, port B, and port D, can set an interrupt on a state change if enabled through registers.

Kamcho includes a total of eight (8) LED drivers, one on each pin of Port E. The brightness control of the LED is accomplished by programmable current sinks. There are two independent LED current sinks controls found in Register 68. The controls are divided into two (2) control groups consisting of the upper nibble and the lower nibble.

The Kamcho device includes a single pin driver, which can supply up to 500mA of current in order to activate an infrared LED for remote control or communications purposes. It is enabled by setting MDIR bit in the **PCONF** register.

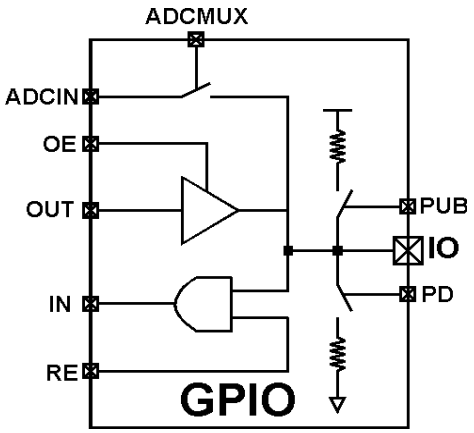


Figure 14 GPIO Pin

11.8.2 GPIO Registers

The following registers control the behavior of the GPIO pins:

Register 40 PORTA input and output register							
PORTA		0x50000060			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
MSB							LSB
Bit7-0 **PA[7:0]**: Port A register bits. 0 = Pin state is '0' 1 = Pin state is '1'							

Register 41 PORTB input and output register							
PORTB		0x50000061			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
MSB							LSB
Bit7-0 **PB[7:0]**: Port B register bits. 0 = Pin state is '0' 1 = Pin state is '1'							

Register 42 PORTC input and output register							
PORTC		0x50000062			0x00		
R/W	Reserved	R/W	R/W	R/W	R/W	R/W	R/W
PC7	-	PC5	PC4	PC3	PC2	PC1	PC0
MSB							LSB

Bit7 **PC[7]**: Port C7 register bits.

0 = Pin state is '0'

1 = Pin state is '1'

Bit5-0 **PC[5:0]**: Port C5-0 register bits.

0 = Pin state is '0'

1 = Pin state is '1'

Register 43 PORTD input and output register							
PORTD		0x50000063			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
MSB							LSB

Bit7-0 **PD[7:0]**: Port D register bits.

0 = Pin state is '0'

1 = Pin state is '1'

Register 44 PORTE input and output register							
PORTE		0x50000064			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0
MSB							LSB
Bit7-0 **PE[7:0]**: Port E register bits. 0 = Pin state is '0' 1 = Pin state is '1'							

Register 45 PORTA output enable register							
PORTAOE		0x50000065			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PAOE7	PAOE6	PAOE5	PAOE4	PAOE3	PAOE2	PAOE1	PAOE0
MSB							LSB
Bit7-0 **PAOE[7:0]**: Port A output enable bits. 0 = Pin is input 1 = Pin is output							

Register 46 PORTB output enable register							
PORTBOE		0x50000066			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PBOE7	PBOE6	PBOE5	PBOE4	PBOE3	PBOE2	PBOE1	PBOE0
MSB							LSB
Bit7-0 **PBOE[7:0]**: Port B output enable bits. 0 = Pin is input 1 = Pin is output							

Register 47 PORTC output enable register							
PORTCOE		0x50000067			0x00		
R/W	Reserved	R/W	R/W	R/W	R/W	R/W	R/W
PCOE7	-	PCOE5	PCOE4	PCOE3	PCOE2	PCOE1	PCOE0
MSB							LSB
Bit7-0 **PCOE[7]**: Port C7 output enable bits. 0 = Pin is input 1 = Pin is output Bit5-0 **PCOE[5:0]**: Port C5-0 output enable bits. 0 = Pin is input 1 = Pin is output							

Register 48 PORTD output enable register							
PORTDOE		0x50000068			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PDOE7	PDOE6	PDOE5	PDOE4	PDOE3	PDOE2	PDOE1	PDOE0
MSB							LSB
Bit7-0 **PDOE[7:0]**: Port D output enable bits. 0 = Pin is input 1 = Pin is output							

Register 49 PORTE output enable register							
PORTEOE		0x50000069			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PEOE7	PEOE6	PEOE5	PEOE4	PEOE3	PEOE2	PEOE1	PEOE0
MSB							LSB
Bit7-0 **PEOE[7:0]**: Port A output enable bits. 0 = Pin is input 1 = Pin is output							

Register 50 PORTA interrupt enable register							
PAINT		0x5000006E			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PAINTE7	PAINTE6	PAINTE5	PAINTE4	PAINTE3	PAINTE2	PAINTE1	PAINTE0
MSB							LSB
Bit7-0 **PAINTE[7:0]**: Port A Interrupt Enable bits 0 = interrupt disabled 1 = interrupt enabled							

Register 51 PORTB interrupt enable register							
PBINT		0x5000006F			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PBINTE7	PBINTE6	PBINTE5	PBINTE4	PBINTE3	PBINTE2	PBINTE1	PBINTE0
MSB							LSB
Bit7-0 **PBINTE[7:0]**: Port B Interrupt Enable bits 0 = interrupt disabled 1 = interrupt enabled							

Register 52 PORTD interrupt enable register							
PDINT		0x50000071			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PDINTE7	PDINTE6	PDINTE5	PDINTE4	PDINTE3	PDINTE2	PDINTE1	PDINTE0
MSB							LSB
Bit7-0 **PDINTE[7:0]**: Port D Interrupt Enable bits 0 = interrupt disabled 1 = interrupt enabled							

Register 53 PORTA pull up enable register							
PAPU		0x50018000			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PAPU7	PAPU6	PAPU5	PAPU4	PAPU3	PAPU2	PAPU1	PAPU0
MSB							LSB
Bit7-0 **PAPU[7:0]**: Port A pull up enable bits (active low) 0 = pull up enabled 1 = pull up disabled							

Register 54 PORTA pull down enable register							
PAPD		0x50018001			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PAPD7	PAPD6	PAPD5	PAPD4	PAPD3	PAPD2	PAPD1	PAPD0
MSB							LSB
Bit7-0 **PAPD[7:0]**: Port A pull down enable bits 0 = pull down disabled 1 = pull down enabled							

Register 55 PORTB pull up enable register							
PBPU		0x50018003			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PBPU7	PBPU6	PBPU5	PBPU4	PBPU3	PBPU2	PBPU1	PBPU0
MSB							LSB
Bit7-0 **PBPU[7:0]**: Port B pull up enable bits (active low) 0 = pull up enabled 1 = pull up disabled							

Register 56 PORTB pull down enable register							
PBPD		0x50018004			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PBPD7	PBPD6	PBPD5	PBPD4	PBPD3	PBPD2	PBPD1	PBPD0
MSB							LSB
Bit7-0 **PBPD[7:0]**: Port B pull down enable bits 0 = pull down disabled 1 = pull down enabled							

Register 57 PORTC pull up enable register							
PCPU		0x50018006			0x00		
R/W	Reserved	R/W	R/W	R/W	R/W	R/W	R/W
PCPU7	-	PCPU5	PCPU4	PCPU3	PCPU2	PCPU1	PCPU0
MSB							LSB
Bit7 **PCPU[7]**: Port C7 pull up enable bits (active low) 0 = pull up enabled 1 = pull up disabled Bit5-0 **PCPU[5:0]**: Port C5-0 pull up enable bits (active low) 0 = pull up enabled 1 = pull up disabled							

Register 58 PORTC pull down enable register							
PCPD		0x50018007			0x00		
R/W	Reserved	R/W	R/W	R/W	R/W	R/W	R/W
PCPD7	-	PCPD5	PCPD4	PCPD3	PCPD2	PCPD1	PCPD0
MSB							LSB
Bit7-0 **PCPD[7]**: Port C7 pull down enable bits 0 = pull down disabled 1 = pull down enabled Bit5-0 **PCPD[5:0]**: Port C5-0 pull down enable bits 0 = pull down disabled 1 = pull down enabled							
Register 59 PORTD pull up enable register							
PDPU		0x50018009			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	Reserved	Reserved
PDPU7	PDPU6	PDPU5	PDPU4	PDPU3	PDPU2	-	-
MSB							LSB
Bit7-2 **PDPU[7:2]**: Port D pull up enable bits (active low) 0 = pull up enabled 1 = pull up disabled **NOTE:** PD0 & PD1 do not have Pull Down nor Pull Up functions. This is due to I2C							
Register 60 PORTD pull down enable register							
PDPD		0x5001800A			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	Reserved	Reserved
PDPD7	PDPD6	PDPD5	PDPD4	PDPD3	PDPD2	-	-
MSB							LSB
Bit7-2 **PDPD[7:2]**: Port D pull down enable bits 0 = pull down disabled 1 = pull down enabled **NOTE:** PD0 & PD1 do not have Pull Down nor Pull Up functions. This is due to I2C							

Register 61 PORTE pull up enable register							
PEPU		0x5001800C			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PEPU7	PEPU6	PEPU5	PEPU4	PEPU3	PEPU2	PEPU1	PEPU0
MSB							LSB
Bit7-0 **PEPU[7:0]**: Port E pull up enable bits (active low) 0 = pull up enabled 1 = pull up disabled							

Register 62 PORTE pull down enable register							
PEPD		0x5001800D			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PEPD7	PEPD6	PEPD5	PEPD4	PEPD3	PEPD2	PEPD1	PEPD0
MSB							LSB
Bit7-0 **PEPD[7:0]**: Port E pull down enable bits 0 = pull down disabled 1 = pull down enabled							

Register 63 PORTA read enable register							
PARE		0x50018002			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PARE7	PARE6	PARE5	PARE4	PARE3	PARE2	PARE1	PARE0
MSB							LSB
Bit7-0 **PARE[7:0]**: Port A read enable bits 0 = Port cannot be read 1 = Port may be read							

Register 64 PORTB read enable register							
PBRE		0x50018005			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PBRE7	PBRE6	PBRE5	PBRE4	PBRE3	PBRE2	PBRE1	PBRE0
MSB							LSB
Bit7-0 **PBRE[7:0]**: Port B read enable bits 0 = Port cannot be read 1 = Port may be read							

Register 65 PORTC read enable register							
PCRE		0x50018008			0x00		
R/W	Reserved	R/W	R/W	R/W	R/W	R/W	R/W
PCRE7	-	PCRE5	PCRE4	PCRE3	PCRE2	PCRE1	PCRE0
MSB							LSB
Bit7-0 **PCRE[7]**: Port C7 read enable bits 0 = Port cannot be read 1 = Port may be read Bit5-0 **PCRE[5:0]**: Port C5-0 read enable bits 0 = Port cannot be read 1 = Port may be read							

Register 66 PORTD read enable register							
PDRE		0x5001800B			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PDRE7	PDRE6	PDRE5	PDRE4	PDRE3	PDRE2	PDRE1	PDRE0
MSB							LSB
Bit7-0 **PDRE[7:0]**: Port D read enable bits 0 = Port cannot be read 1 = Port may be read							

Register 67 PORTE read enable register							
PERE		0x5001800E			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PERE7	PERE6	PERE5	PERE4	PERE3	PERE2	PERE1	PERE0
MSB							LSB
Bit7-0 **PERE[7:0]**: Port E read enable bits 0 = Port cannot be read 1 = Port may be read							

Register 68 PORTE LED driver configuration register							
PELEDCFG		0x5000006B			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
LEDMD7	LEDMD6	LEDMD5	LEDMD4	LEDMD3	LEDMD2	LEDMD1	LEDMD0
MSB							LSB
Bit7-0 **LEDMD[7:0]**: Enable Port E LED driver 0 = GPIO mode 1 = LED mode							

Register 69 PORTE LED trim register							
PELEDTRIM		0x5001800F			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PE74IS3	PE74IS2	PE74IS1	PE74IS0	PE30IS3	PE30IS2	PE30IS1	PE30IS0
MSB							LSB
Bit7-4 **PE74IS[3:0]**: Port E upper nibble LED trim current simultaneously controls current to LEDs connected to PE7-4, from 0 to 30mA in 2mA step. Bit3-0 **PE30IS[3:0]**: Port E lower nibble LED trim current simultaneously controls current to LEDs connected to PE3-0, from 0 to 30mA in 2mA step **NOTE:** The LED control is controlled in two Groups: Upper Nibble and Lower Nibble							

Register 70 Pin configuration register							
PCONF		0x5000006A			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
I2C_RT1	I2C_RT0	UPSWAP	UTXPOL	MDSPI	MDI2C	MDUART	MDIR
MSB							LSB

Bit7-6 **I2C_RT[1:0]**: I2C Resistor Trim

00 = Open

01 = 1k

10 = 10k

11 = 100k

Bit5 **UPSWAP**: UART Pins Swap Bit.

0 = Pins are not swapped (TX=PD[7] and RX = PD[6])

1 = Pins are swapped (TX=PD[6] and RX = PD[7])

Bit4 **UTXPOL**: UART signals polarity.

0 = normal polarity

1 = inverted polarity

Bit3 **MDSPI**: SPI mode enable

0 = GPIO

1 = SPI mode

Bit2 **MDI2C**: I2C mode enable

0 = GPIO

1 = I2C mode

Bit1 **MDUART**: UART mode enable

0 = GPIO

1 = UART mode

Bit0 **MDIR**: Enable IR LED driver

0 = IR LED driver disabled (PC7 in GPIO mode)

1 = IR LED driver enabled on pin PC7

Register 71 PORTA PWM1 configuration register							
PAPWMCFG		0x5000006C			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PAPWM17	PAPWM16	PAPWM15	PAPWM14	PAPWM13	PAPWM12	PAPWM11	PAPWM10
MSB							LSB
Bit7-0 **PAPWM1[7:0]**: Connect Port A to PWM1 0 = GPIO mode 1 = PWM1 mode							

Register 72 PORTC PWM2 configuration register							
PCPWMCFG		0x5000006D			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PCPWM27	PCPWM26	PCPWM25	PCPWM24	PCPWM23	PCPWM22	PCPWM21	PCPWM20
MSB							LSB
Bit7-0 **PCPWM2[7:0]**: Connect Port C to PWM2 0 = GPIO mode 1 = PWM2 mode							

11.9 Clock Sources

Kamcho provides three clock sources:

- Internal auxiliary oscillator running at 10 kHz. This oscillator is always running, even when the part is in sleep mode.
- Internal RC oscillator running at 12MHz. After a reset Kamcho starts with this oscillator selected. From this point on the user may select to enable and select the crystal or the auxiliary oscillators.
- Real Time Clock Crystal oscillator (Typically 32.768kHz)

The clock may be divided from the oscillator. The division ratio is controlled via a register setting.

11.9.1 Clock Sources Characteristics

The following table defines the main characteristics of the clock sources:

Table 21 Clock Performance Specification

name	conditions	min	typ	max	unit
Crystal Oscillator frequency			32.768		kHz
Frequency stability	Using defined crystal			TBD	ppm
Current Consumption Crystal			0.5	1	µA
Sleep Current Consumption Crystal	Crystal oscillator disabled			100	nA
Auxiliary Oscillator			10		kHz
Auxiliary Oscillator accuracy	T_A=27°C			25	%
Auxiliary Oscillator current consumption	Enabled		0.5	1	µA
Main RC Oscillator frequency			12		MHz
Main RC Oscillator accuracy	T_A=27°C. After factory calibration			1	%
Main RC Oscillator Current Consumption	Enabled		300	500	µA

11.9.2 Real Time Clock Operation

The 32.768kHz crystal oscillator is employed to generate a real time clock (RTC) function. The oscillator drives a counter/divider, which may be programmed to produce “ticks” at several discrete intervals. Each tick may be used to set a microcontroller interrupt. The operation is defined below.

11.9.3 Clock Related Registers

The following registers are used to control the behavior of the clock sources:

Register 73 Clock Control Register							
PMUCLK		0x50000000			0x15		
R/W	R/W	R	R	R/W	R/W	R/W	R/W
CKD1	CKD0	XOMON	RCMON	XO_CK_ENB	RC_CK_ENB	CKSEL1	CKSEL0
MSB							LSB

Bit7-6 **CKD[1:0]**: Clock Frequency Divider

00 = Clock Divided by 1

01 = Clock Divided by 2

10 = Clock Divided by 4

11 = Clock Divided by 8

Bit5 **XOMON**: Crystal Oscillator Monitor

0 = Crystal Oscillator Inactive / 1 = Crystal Oscillator Active

Bit4 **RCMON**: RC Oscillator Monitor

0 = RC Oscillator Inactive / 1 = RC Oscillator Active

Bit3 **XO_CK_ENB**: Crystal Oscillator Control

0 = Crystal Oscillator Disable / 1 = Crystal Oscillator Enable

Bit2 **RC_CK_ENB**: RC Oscillator Control

0 = RC Oscillator Disable / 1 = RC Oscillator Enable

Bit1-0 **CKSEL[1:0]**: Clock Select

00 = 10 kHz Auxiliary Clock

01 = 12MHz RC Oscillator Clock

10 = Crystal Oscillator Clock divided by 2

11 = Clock fault indicator → Change to 12MHz RC oscillator*

*Note: This is the clock selected after Power-On-Reset.

Register 74 RTC Control Register							
RTCCTRL		0x5001801A			0x00		
Reserved	Reserved	Reserved	Reserved	R/W	R/W	R/W	R/W
-	-	-	-	RTCDIV1	RTCDIV0	RTCTRM1	RTCTRM0
MSB							LSB

Bit3-2 **RTCDIV[1:0]**: Real Time Clock Divider

00 = 128

01 = 2048

10 = 32768

11 = 524288

Bit1-0 **RTCTRM[1:0]**: Real Time Clock Bias Resistor Trim

00 = 150k

01 = 200k

10 = 250k

11 = 350k

Register 75 10kHz RC Trim Register							
RCC10KTRIM		0x50018013			0xF0		
Reserved	Reserved	Reserved	R/W	R/W	R/W	R/W	R/W
-	-	-	RC10KTRIM 4	RC10KTRIM 3	RC10KTRIM 2	RC10KTRIM 1	RC10KTRIM 0
MSB							LSB

Bit4-0 **RC10KTRIM[4:0]**: 10kHz RC oscillator Trim

00 = highest frequency

1F = lowest frequency

Register 76 12MHz RC oscillator Trim0 Register							
RCTRIM0		0x50018014			0x57		
R/W	R/W	R/W	Reserved	Reserved	Reserved	Reserved	Reserved
RC12MCCAL2	RC12MCCAL1	RC12MCCAL0	-	-	-	-	-
MSB							LSB
Bit7-5 **RC12MCAL[2:0]**: Trim RC oscillator capacitor							

Register 77 12MHz RC oscillator Trim1 Register							
RCTRIM1		0x50018015			0x8A		
Reserved	Reserved	Reserved	Reserved	R/W	R/W	R/W	R/W
-	-	-	-	RC12MCCAL6	RC12MCCAL5	RC12MCCAL4	RC12MCCAL3
MSB							LSB
Bit6-0 **RC12MCAL[6:0]**: Trim RC oscillator capacitor (Register 77 contains 3 LSBs) 7F = Lowest frequency 00 = Highest frequency							

11.9.4 Clock Sources Usage Description

Upon Reset or Power-On Reset the system starts using the internal 12MHz RC oscillator.

Depending on the application requirements the designer can:

- Enable or disable the internal RC oscillator
- Enable or disable the external crystal
- Select the system clock source: 12MHz RC, 10kHz Auxiliary or Crystal
- Enable the clock monitor interrupt to detect and process eventual failures in the crystal clock mode

Example Code: Enable the Crystal oscillator. Wait for it to be stable and then selects it. Also enables the clock monitor interrupt and create an interrupt handler routine for it.

```
CLK_CrystalControl( XTON );       //Enable Crystal Clock
for ( i = 0; i < 20000; i++);           //Some delay to allow for crystal to start
while ( CLK_CrystalMonitor () == 0 ); //Check that the crystal is OK
CLK_SelectClockSource( XTCLOCK );     //Selects crystal as clock source
NVIC_EnableIRQ(BrownOut_IRQn);          //Enable the clock monitor interrupt
```

Related Interrupt handler for the clock monitor:

```
void ClkMon_Handler ( void )
{
  If (CLK_CrystalMonitor()==0)
  {
          //Here add the code to handle the failure of the crystal clock
  }
  If (CLK_RcMonitor()==0)
  {
          //Here add the code to handle the failure of the RC clock
  }
}
```

11.10 Power Management Unit (PMU)

Kamcho implements a power management unit. Its main characteristics are:

- HW reset - Affects all aspects of Kamcho
- SW reset - Does not affect clock nor brownout setup
- Selectable Sleep mode and Deep Sleep (Halt) Mode
- Programmable brownout detector

11.10.1 PMU Registers

Register 78 Processor Control Register							
PMURST		0x50000001			0x01		
W	W	R/W	Reserved	Reserved	Reserved	R	R/W
HWRST	SWRST	DLEEP	-	-	-	BROUT	PORF
MSB							LSB

Bit7 **HWRST**: Hardware reset

0 = Idle 1 = Hardware reset (automatically cleared after reset process completed)

Bit6 **SWRST**: Software reset

0 = Idle 1 = Software reset (automatically cleared after reset process completed)

Bit5 **DLEEP**: Deep sleep (HALT) mode

Writing:

0 = Clear deep sleep flag 1 = Put the system in deep sleep - Halt

Reading:

0 = Flag cleared 1 = system in deep sleep mode

Bit1 **BROUT**: Brownout indicator

0 = No brownout 1 = Brownout

Bit0 **PORF**: Power-On Reset flag

Writing:

0 = Clear POR 1 = No effect

Reading:

0 = POR flag already cleared by application 1 = The system just came out of POR or HW Reset

Register 79 Brown Out Reset Control Register							
PMUBOR		0x50000002			0x88		
R/W	Reserved	Reserved	Reserved	R/W	R/W	R/W	R/W
BOREN	-	-	-	BORRST	BORINT	BOUTVALUE1	BOUTVALUE0
MSB							LSB

Bit7 **BOREN**: Brownout enable

0 = Brownout disabled

1 = Brownout enabled

Bit3 **BORRST**: Brownout Reset enable

0 = Disable Brownout based reset

1 = Enable Brownout based reset

Bit2 **BORINT**: Brownout interrupt

1 = Brownout interrupt disabled

0 = Brownout interrupt enabled

Bit1-0 **BOUTVALUE [1:0]**: Brownout threshold value

00 = 2.0V

01 = 2.2V

10 = 2.4V

11 = 2.6V

Register 80 Power Configuration Register							
PMUCONF		0x50000005			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
SERFAST	BATOP	RTSELA2	RTSELA1	RTSELA0	RTSELB2	RTSELB1	RTSELB0
MSB							LSB

Bit7 **SERFAST**: Serial Interface Clock Mode

0 = divided clock rate

1 = full clock rate

Bit6 **BATOP**: Battery Operation Mode

0 = Regulator as power source

1 = Battery as power source

Bit5-3 **RTSELA[2:0]**: Test Output A (PD[0]) Select

001 = 10kHz RC oscillator output

010 = 12MHz RC oscillator output

011 = real time clock oscillator output divided by 2

100 = divided real time clock output (see 0)

110 = Brown out signal

Bit2-0 **RTSELB [2:0]**: Test Output B (PD[1]) Select

001 = full rate system clock

010 = divided system clock (see 0)

011 = serial clock

100 = real time ADC output

Register 81 Power Configuration2 Register							
PMUCONF2		0x5001801B			0x0F		
R/W	Reserved	Reserved	Reserved	R/W	R/W	R/W	R/W
	-	-	-	VF_TRIM3	VF_TRIM2	VF_TRIM1	VF_TRIM0
MSB							LSB
Bit3-0 **VF_TRIM[3:0]**: Trim setting for Flash LDO							

11.10.2 PMU Usage Description

The PMU module allows for the control of reset, deep sleep (halt), sleep, and brownout.

11.10.2.1 PMU control of Reset:

There are two forms of reset that can be issued:

- Hardware reset: In this reset all peripherals are reset, the 10kHz clock is selected and all other clock sources are disabled, but the brownout selection is kept.
- Software reset: In this reset all peripherals are reset but the clock setup is kept unchanged along with the brownout selection.

Code Examples: HW reset and SW reset:

```
PMU_HwReset();

PMU_SwReset();
```

11.10.2.2 PMU control of sleep and deep sleep (halt) modes:

The PMU can set the system into sleep or deep sleep (halt) modes.

11.10.2.2.1 Deep Sleep (halt) mode

In the deep sleep (halt) mode:

- The CPU is halted and its power supply disabled
- Any enabled clock source will continue to operate
- The three timers (Timer0, Timer1 and Timer2) and the SysTick Timer will stop operating
- All other peripherals will keep running (if enabled and fed by their required source clock)
- The system will leave the deep sleep (halt) mode only through a reset or POR (**P**ower-**O**n **R**eset). The sources of a reset can be the wakeup timer or any peripheral that generates an interrupt independently of the interrupt being enabled by the NVIC module (**N**ested **V**ector **I**nterrupt **C**ontroller)

Note: For those peripherals that have a bit in their registers that locally enables an interrupt, this register has to be enabled in order to reset the system. Example: For the GPIOs ports PORTA, PORTB, and PORTD the INTE bits of the I/O pins selected to reset the part upon change must be set.

Code Example: Enabling the reset by enabling an interrupt from PORTA[0] upon change. (Port A, bit0)

```
//PortA.0 interrupt enabled, pull up enabled
Port_Config(PTA, 0, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00);
        PMU_Deep_Sleep();       //System in deep sleep (halt)
```

11.10.2.2.2 Sleep Mode

In the sleep mode:

- the CPU is halted
- Any enabled clock source will continue to operate
- All timers (Timer0, Timer1 and Timer2) and the SysTick Timer will continue operating
- All other peripherals will keep running if enabled and fed by their required source clock
- Besides a POR and/or reset, the system will leave the sleep mode also through an interrupt; the sources of an interrupt can be any peripheral generating an interrupt.

Note: The interrupt must be enabled by the NVIC module.(**N**ested **V**ector **I**nterrupt **C**ontroller) and for those peripherals that have in their registers a bit that locally enables the interrupt this register also has to be enabled in order to generate an interrupt and wakeup the system from sleep.

Example: For the GPIOs ports PORTA, PORTB and PORTC the INTE bits of the I/O pins selected to reset the part upon change must be set.

Code Example:

```
//PortA.0 interrupt enabled, pull up enabled
Port_Config(PTA, 0, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00);
NVIC_EnableIRQ(PIN_IRQn);      //Pin change interrupt enabled in NVIC
        PMU_Sleep();                    //Part in sleep mode
```

11.10.2.2.3 PMU control of Brown Out Reset:

The PMU controls the Brown Out Reset. It entails:

- Enabling or disabling the Brown Out Reset circuit
- Selecting the behavior when a Brown Out is detected:
    - Generate an interrupt
    - Reset the system
- Selecting the brownout voltage level

Code Example: Enable the BOR, with interrupt but no reset and with 2.4V level.

```
        BOR_ResetControl(BORRSTDIS); //Brownout reset disabled
        BOR_IntControl(BORINTEN);     //Brownout interrupt enabled
        BOR_Level(BOR24V);            //Brownout level = 2.4V
        BOR_Control(BOREN);           //Brownout enabled
```

11.11 Wake-Up Timer

In addition to the Timer0/1/2, Kamcho includes a timer capable of waking-up the microcontroller from a deep sleep (halt) state.

The wake up timer is a timer used to allow for recovery from deep sleep (halt), including when the microcontroller is disconnected from its power supply.

The following register controls the wake-up timer:

Register 82		**Wakeup Timer Control**					
WKPTIME		0x50000004			0x00		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
MANT3	MANT2	MANT1	MANT0	EXP3	EXP2	EXP1	EXP0
MSB							LSB

Bit7-4 **MANT [3:0]**: Mantissa of the wakeup timer

Bit3-0 **EXP [3:0]**: Exponent of the wakeup timer (range: 0...12)

$$WakeupPeriod = \frac{2 * Mantissa * 2^{Exponent}}{SystemClock}$$

For instance, a value of 0x54 would give a time of: (Assuming the application is running from the 10kHz internal oscillator)

$$WakeupPeriod = \frac{2 * 5 * 2^4}{10kHz} = 16m\sec$$

Code Example: Enabling the wakeup timer according to the previous example.

WKP_Timing(5,4) //Select the mantissa=5 and exponent=4

```
PMU_Deep_Sleep(SLEEPON); //Put the part in deep sleep mode, it will
                         //Reset in 16msec.
```

12 PACKAGE OUTLINE

The dimensions of the package are defined in the following table and drawings:

		SYMBOL	MIN	NOM	MAX
TOTAL THICKNESS		A	0.8	0.85	0.9
STAND OFF		A1	0	0.035	0.05
MOLD THICKNESS		A2	---	0.65	0.67
L/F THICKNESS		A3		0.203 REF	
LEAD WIDTH		b	0.16	0.21	0.26
BODY SIZE	X	D		7 BSC	
	Y	E		7 BSC	
LEAD PITCH		e		0.5 BSC	
EP SIZE	X	J	4.4	4.5	4.6
	Y	K	4.4	4.5	4.6
LEAD LENGTH		L	0.35	0.4	0.45
		R	1.7	1.8	1.9
PACKAGE EDGE TOLERANCE		aaa		0.1	
MOLD FLATNESS		bbb		0.1	
COPLANARITY		ccc		0.08	
LEAD OFFSET		ddd		0.1	
EXPOSED PAD OFFSET		eee		0.1	

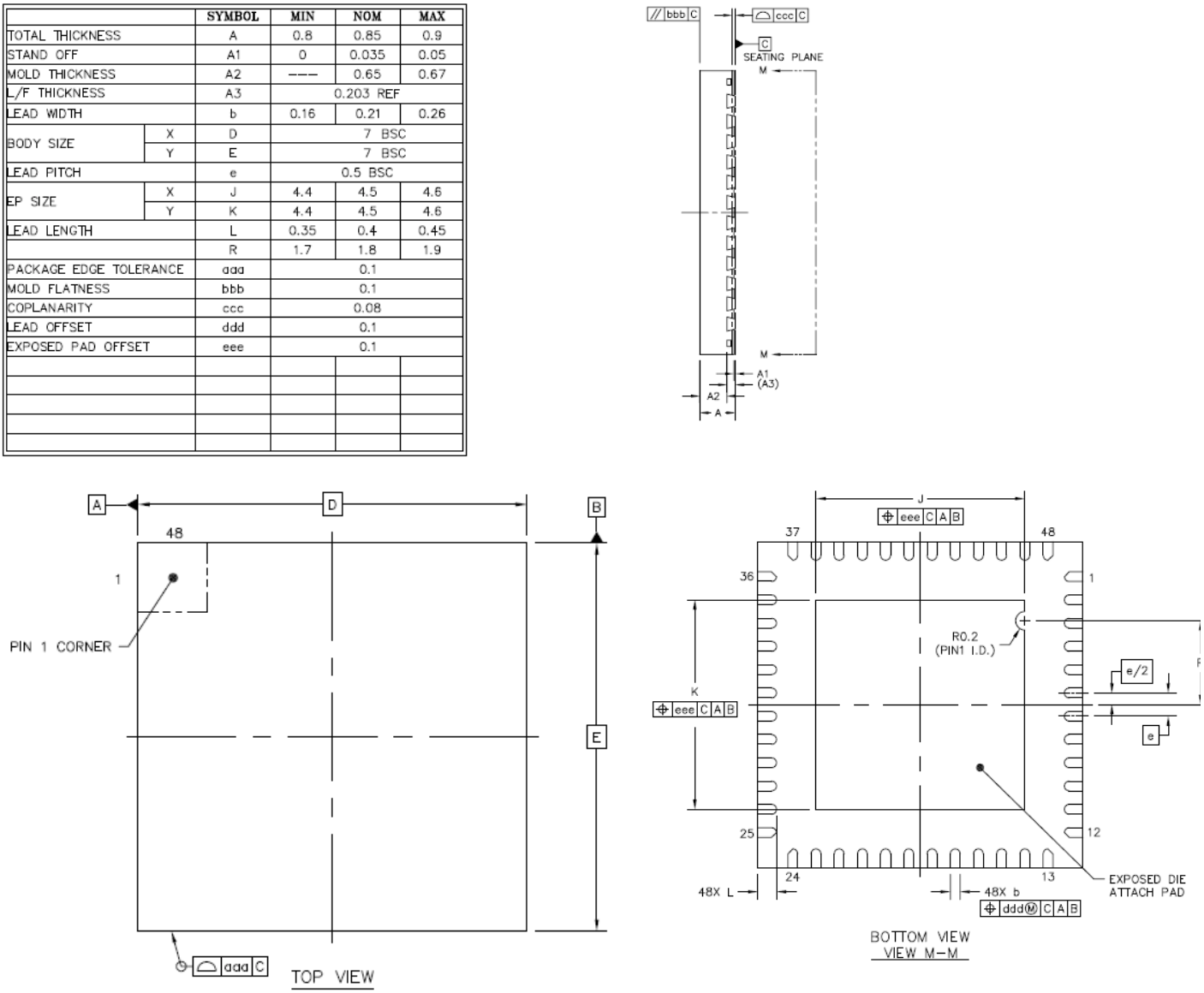


Figure 15 QFN (7mm x 7mm) 48-pin Package Dimensions

13 EVALUATION BOARD

The indie semi-conductor iND81205 "Kamcho" general purpose microcontroller device can be tested using an evaluation kit called Kamcho mini. The board allows the user to upload software via an ARM proprietary interface known as "SerialWire", and it provides access to all peripheral pins via an edge connector, which may be used to connect to test equipment or to drive additional application circuitry.

Information to order the board can be found on indie semiconductor website with all Software Development Kit and documentation. See references [3] and [4]

13.1 Kamcho mini development kit board top view

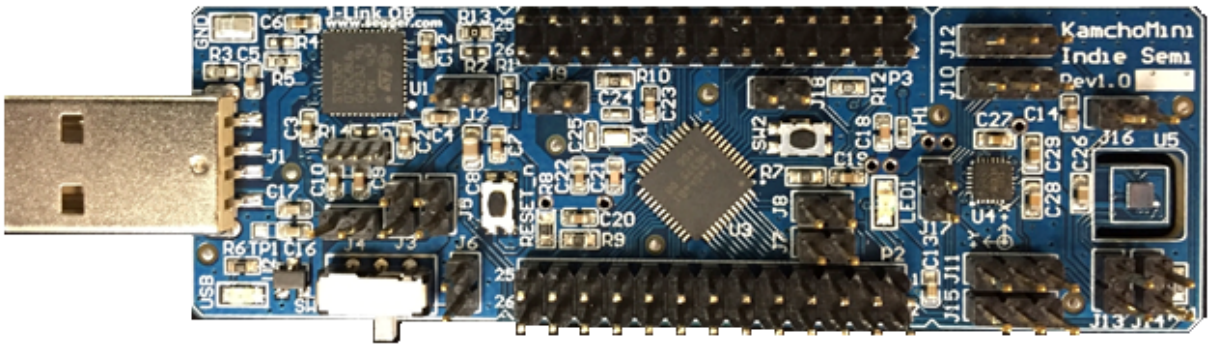


Figure 16 Kamcho Mini evaluation kit

13.2 Kamcho development board scematic

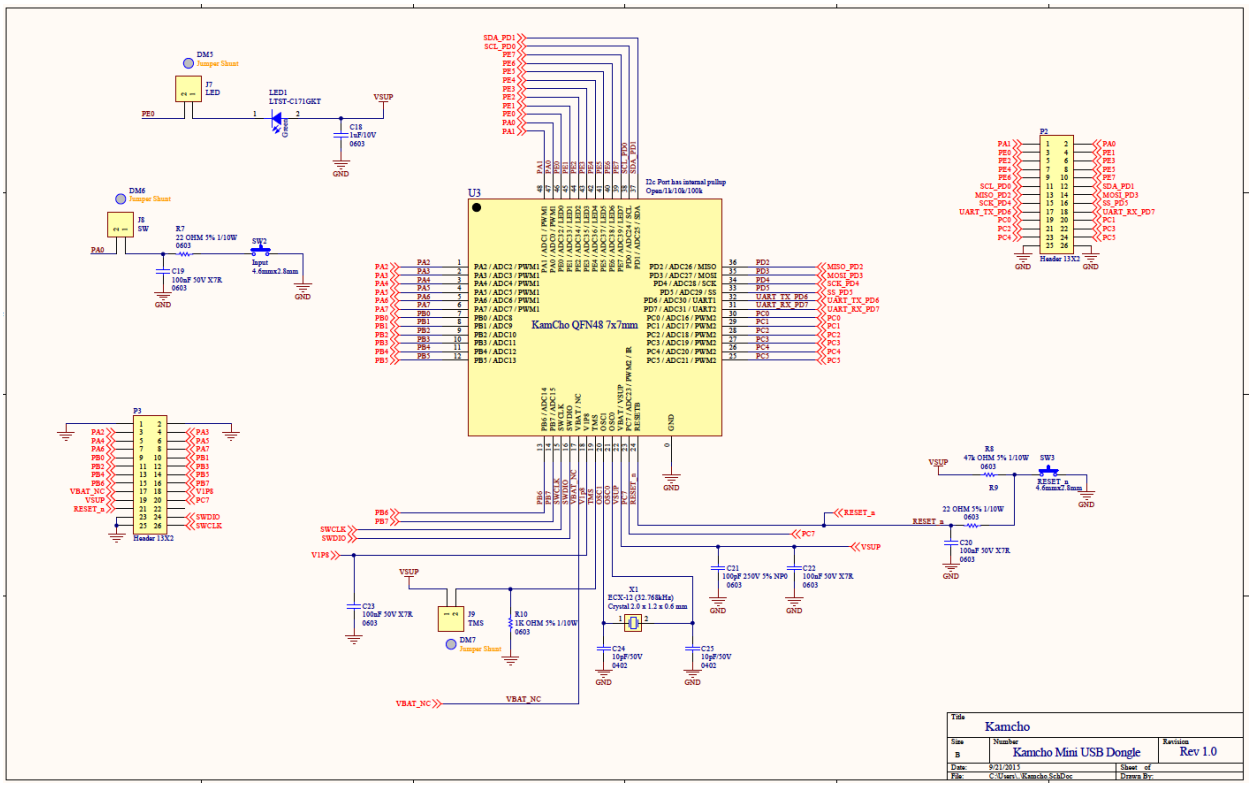


Figure 17 Kamcho Mini evaluation kit schematic (extract)

14 DOCUMENT REVISION HISTORY

Table 22 Revision History

Rev #	Date	Description	By
1.0	4 May 2012	Kamcho_B1 Updates	DDK
2.0	04/02/2015	Indie format datasheet from AyDeeKay– 11/11/14 (v1.31 - JEA)	CR
2.1	10/02/2015	Indie new contact address and minor correction	CR
2.2	10/30/2015	DDK corrections and EVKIt info added	DDK/CR
2.3	11/04/2015	Minor typos fixes	CR

15 REFERENCES

[1] "**AyDeeKay_Core_160_8.pdf"**: specification of indie product CLOUGH "ARM™ core M0"

[2] http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

[3] indie Kamcho_mini_starting_guide_R1015.pdf

[4] indie Kamcho_mini_board_schematic_Rev1.0.pdf

16 CONTACTS AND ORDERING INFORMATION

Table 23 Ordering information

Order code	Description	Package(1)
iND81205	Kamcho microcontroller	7x7 QFN
Kamcho EVKIT I	Kamcho demo board	
Kamcho_mini EVKIT	Kamcho evaluation kit	

1. Back-Side Coating: It is a tape of 40 micron thickness, used for protecting and reinforcing the chip surface.

United States
32 Journey
Aliso Viejo, California 92656, USA
Tel: +1 949-608-0854
sales@indiesemi.com

China
232 Room, Donghai Wanhao Plaza,
South Hi-tech 11th Road, Hi-tech Industry Park,
Nanshan District, Shenzhen, China.
Tel: +86 755-86116939

Scotland
MWB Business Exchange
9-10 St. Andrew Square
Edinburgh EH2 2AF, Scotland
Tel: +44 131 718 6378

http://www.indiesemi.com/

Important Notice

indie semiconductor reserves the right to make changes, corrections, enhancements, modifications, and improvements to indie semiconductor products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on indie semiconductor products before placing orders. indie semiconductor products are sold pursuant to indie semiconductor's terms and conditions of sale in place at the time of order acknowledgement. Purchasers are solely responsible for the choice, selection, and use of indie semiconductor products and services described herein. indie semiconductor assumes no liability for the choice, selection, application assistance or the design of Purchasers' products.

No license, expressed or implied, to any intellectual property right is granted by indie semiconductor by this document.

The materials, products and information are provided "as is" without warranty of any kind, whether express, implied, statutory, or otherwise, including fitness for a particular purpose or use, merchantability, performance, quality or non-infringement of any intellectual property right. Indie semiconductor does not warrant the accuracy or completeness of the information, text, graphics or other items contained herein. indie semiconductor shall not be liable for any damages, including but not limited to any special, indirect, incidental, statutory, or consequential damages, including without limitation, lost of revenues or lost profits that may result from the use of the materials or information , whether or not the recipient of material has been advised of the possibility of such damage.

Unless expressly approved in writing by two authorized indie semiconductor representatives, indie semiconductor products are not designed, intended, warranted, or authorized for use as components in military, space, or aircraft, in systems intended to support or sustain life, or for any other application in which the failure or malfunction of the indie semiconductor product may result in personal injury, death, or severe property or environmental damage.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.