PDK82C13 / PDK82C13-D
ADC-Type Enhanced
Field Programmable Processor Array
(FPPA™)

Patent Pending

Data Sheet

Preliminary

Version 0.10 – Oct 31, 2007

10F-2, No. 1, Sec. 2, Dong-Da Road, Hsin-Chu 300, Taiwan, R.O.C.

TEL: 886-3-532-7598 www.padauk.com.tw

IMPORTANT NOTICE

PADAUK Technology reserves the right to make changes to its products or to terminate production of its products at any time without notice. Customers are strongly recommended to contact PADAUK Technology for the latest information and verify whether the information is correct and complete before placing orders.

PADAUK Technology products are not warranted to be suitable for use in life-support applications or other critical applications. PADAUK Technology assumes no liability for such applications. Critical applications include, but are not limited to, those which may involve potential risks of death, personal injury, fire or severe property damage.

PADAUK Technology assumes no responsibility for any issue caused by a customer's product design. Customers should design and verify their products within the ranges guaranteed by PADAUK Technology. In order to minimize the risks in customers' products, customers should design a product with adequate operating safeguards.

Table of Contents

Revision History:

Revision	Date	Description
0.10	2007/10/31	1st version

Features

High Performance RISC CPU Array

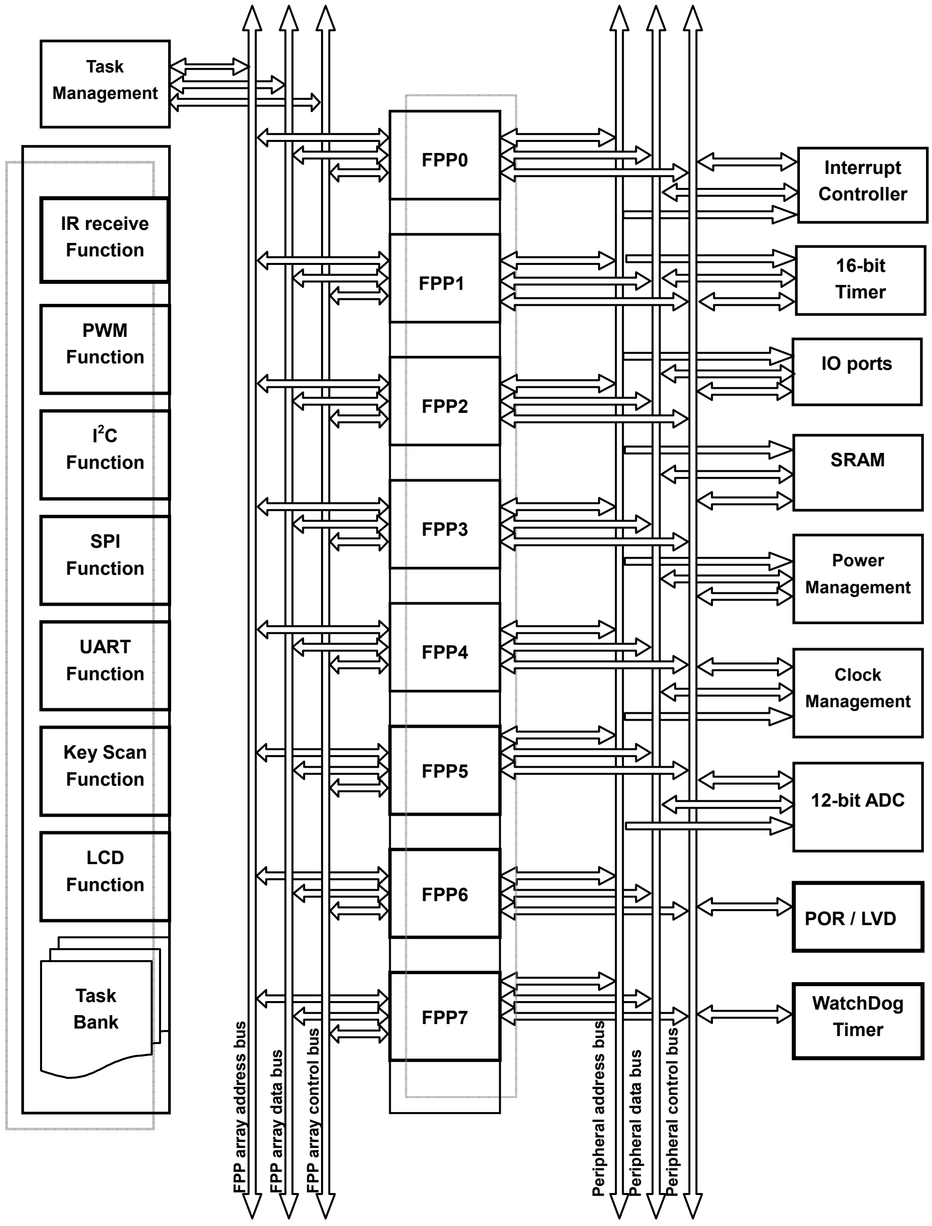
- Patent Pending Field Programmable Processor Array (FPPA™) Technology
- 8x8 processor array with parallel processing capability
- 2KW OTP program memory for all FPP units
- 192 Bytes data RAM for all FPP units
- 102 powerful instructions
- All instructions are 1T except indirect memory access
- One cycle for branch instructions to reduce overhead
- Programmable stack pointer / adjustable stack level
- Direct / indirect addressing modes for data and instructions
- Bit-manipulation instructions
- All data memories are available for use as an index pointer
- Support security function to protect OTP data
- Separated IO space and memory space
- Powerful instructions for peripheral functions
- Powerful instructions for intra-FPP handshaking

System Functions

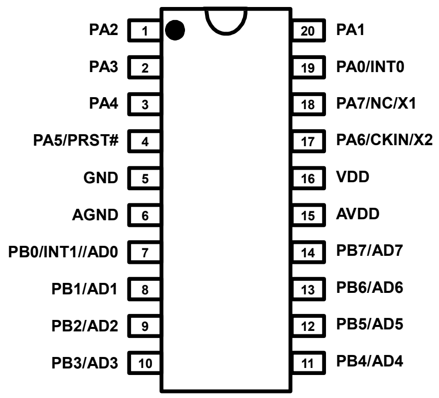
- Clock modes: internal high RC, internal low RC, external RC, external crystal and external clock
- Built-in Power On Reset and Low Voltage Detector
- Built-in internal high RC oscillator
- One hardware 16-bit timer
- Maximum 8-channel 12-bit ADC
- Support software full duplex UART
- Support software flexible PWM waveform generation
- Support software SPI serial protocol
- 20-pin SSOP / DIP Package
- 15 IO pins and 1 input pin
- IO pins with 15mA capability
- Serial in-system programming
- Operating voltage range
  - f_{SYS}= 16MHz@5.0V
  - f_{SYS}= 8MHz@3.3V
- Maximum performance
  - Crystal mode: 16MIPS@VDD=5.0V
  - External RC Mode: 8MIPS@VDD=5.0V
- Operating voltage range: 2.5V ~ 5.5V
- Operating temperature range: -40°C ~ 105°C
- Operating frequency range
  - Crystal mode:
    - DC ~ 16MHz@VDD=5.0V
    - DC ~ 8MHz@VDD=3.3V
  - External RC Mode:
    - DC ~ 8MHz@VDD=5.0V
    - DC ~ 4MHz@VDD=3.3V
- Low power consumption
  - $I_{operating}$ ~ 1.2mA@1MIPS / VDD=5.0V
  - $I_{operating}$ ~ 9uA@32KHz / VDD=3.3V
  - $I_{standby}$ ~ 1.0uA@VDD=5.0V
  - $I_{standby}$ ~ 0.4uA@VDD=3.3V

General Description and Block Diagram

The PDK82C13 is an ADC-Type of PADAUK's parallel processing, fully static, OTP-based CMOS 8x8 processor array that can execute numerous peripheral functions in parallel. It employs RISC architecture based on patent pending FPPA™ (Field Programmable Processor Array) technology and all the instructions are executed in one cycle except that some instructions are two cycles that handle indirect memory access. One up to 8 channels 12-bit ADC is also built inside the chip. By using FPPA™ technology, it allows most of peripheral functions to be performed by software to meet customers' requirements in different applications. The parallel processing architecture provides a system true real-time multi-tasking capability by hardware approach.



Pin Assignment and Pin Description



PDK82C13 (SSOP20-150mil)
PDK82C13-D (DIP20-300mil)

Pin Description for PDK82C13 / PDK82C13-D

Pin No.	Pin Name	Description
18	PA7/NC/X1	This pin can be used as (1) Bit 7 of port A when an internal RC oscillator is used and can be configured as input/output, with pull-up resistor, open-drain output mode by software. (2) Leave this pin no connection when an external clock oscillator is used. (3) X1 when a crystal oscillator or an external RC oscillator is used.
17	PA6/CKIN/X2	This pin can be used as (1) Bit 6 of port A when an external crystal oscillator is not used and can be configured as input/output, with pull-up resistor, open-drain output mode by software. (2) Clock input when an external clock oscillator is used. (3) X2 when a crystal oscillator is used.
4	PA5/PRST#	This input pin can be used as (1) hardware reset of this chip. (2) Bit 5 of port A. Please note that this pin is for input only and does not have pull-up or pull-down resistor.
3 2 1 20	PA4 PA3 PA2 PA1	Bit 4, 3, 2, and 1 of port A. These four pins can each be configured as input/output, with pull-up resistor, open-drain output mode by software.
19	PA0/INT0	Bit 0 of port A or external interrupt line 0. This pin can be configured as input/output, with pull-up resistor, open-drain output mode by software and can be used as an external interrupt line 0. Both rising edge and falling edge are accepted to request interrupt service.
14 13 12 11 10	PB7/AD7 PB6/AD6 PB5/AD5 PB4/AD4 PB3/AD3	Bit 7~0 of port B or channel 7~0 of analog input. These eight pins can each be configured as analog input, digital input, two-states output mode with pull-up resistor independently by software and PB0/AD0/INT1 can be used as an external interrupt line, both rising edge and falling edge are accepted to request interrupt service. When any of these eight pins acts as analog inputs, it must be programmed as analog input via analog input control register to avoid leakage current.

9	PB2/AD2	
8	PB1/AD1	
7	PB0/AD0/INT1	
16	VDD	Digital Positive power
15	AVDD	Analog Positive Power
5	GND	Digital Ground
6	AGND	Analog Ground

Device Characteristics

DC Characteristics

Symbol	Description	Min	Typ	Max	Unit	Conditions (Ta=25℃)
V_{DD}	Operating Voltage	2.5	5.0	5.5	V	
I_{OP}	Operating Current		1.2 12 9		mA mA uA	f_{SYS}=1MIPS@5.0V f_{SYS}=16MIPS@5.0V f_{SYS}=32KHz@3.3V
I_{PD}	Power Down Current		1.0 0.4		uA uA	f_{SYS}= 0Hz,VDD=5.0V f_{SYS}= 0Hz,VDD=3.0V
V_{IL}	Input low voltage for IO lines	0		0.3V_{DD}	V	
V_{IH}	Input high voltage for IO lines	0.7 V_{DD}		V_{DD}	V	
I_{OL}	IO lines sink current		15		mA	V_{DD}=5.0V, V_{OL}=0.5V
I_{OH}	IO lines drive current		-15		mA	V_{DD}=5.0V, V_{OH}=4.5V
R_{PH}	Pull-high Resistance		80		KΩ	V_{DD}=5.0V
V_{AD}	AD Input Voltage	0		VDD	V	
AD DNL	AD Differential NonLinearity		±2*		LSB	
AD INL	AD Integral NonLinearity		±3*		LSB	

*These parameters are for design reference, not tested for each chip.

AC Characteristics

Symbol	Description	Min	Typ	Max	Unit	Conditions
f_{SYS}	System clock crystal oscillator external RC oscillator internal high RC oscillator internal low RC oscillator	 0 0 	 32K	 16M 8M 16M 	 Hz 	 V_{DD} = 5.0V V_{DD} = 5.0V V_{DD} = 5.0V V_{DD} = 5.0V
t_{WDT}	Watchdog timeout period	1024*(1 / f_{ILRC}), where f_{ILRC} is the frequency of the internal low RC oscillator (Note: f_{ILRC} will drift with temperature and voltage)				
t_{SBP}	System boot-up period	2048*(1 / f_{ILRC}), where f_{ILRC} is the frequency of the internal low RC oscillator (Note: f_{ILRC} will drift with temperature and voltage)				
t_{INT}	Interrupt pulse width	30			ns	V_{DD} = 5.0V
t_{RST}	External reset pulse width	Minimum is 4*(1/f_{ILRC}), where f_{ILRC} is the frequency of the internal low RC oscillator				

Absolute Maximum Ratings

- Supply Voltage .... 2.5V ~ 5.5V
- Input Voltage .... -0.3V ~ VDD + 0.3V
- Operating Temperature .... -40℃ ~ 105℃
- Storage Temperature .... -50℃ ~ 125℃

Functional Description

Processing Units

There are 8 processing units (FPP unit) inside the chip of PDK82C13. In each processing unit, it includes (i) its own Program Counter to control the program execution sequence (ii) its own Stack Pointer to store or restore the program counter for program execution (iii) its own accumulator (iv) Status Flag to record the status of program execution.

The FPP unit can be enabled or disabled by programming the FPP unit Enable Register, only FPP0 is enabled after power-on reset. The system initialization will be started from FPP0 and other FPP unit can be enabled by user's program if necessary. All the FPP units can be enabled or disabled by using any one FPP unit

Program Counter

Program Counter (PC) is the unit that contains the address of an instruction to be executed next. The program counter is automatically incremented at each instruction cycle so that instructions are retrieved sequentially from the program memory. Certain instructions, such as branches and subroutine calls, interrupt the sequence by placing a new value in the program counter. The bit length of the program counter is 11 for PDK82C13. The program counter of FPP0 is 0 after hardware reset, 1 for FPP1, 2 for FPP2, and so on. Whenever an interrupt happens, the program counter will jump to 'h10 for interrupt service routine. Each FPP unit has its own program counter to control the program execution sequence. One FPP unit can read or control the program counter of another FPP unit by using *pushw* / *popw* instructions.

Program Memory -- OTP

The OTP (One Time Programmable) program memory is used to store the program instructions to be executed. All program codes for every FPP unit are stored in this OTP, regardless which FPP unit the program code belongs to. The OTP program memory may contains the data, tables and interrupt entry. The OTP program memory for PDK82C13 is 2KW that is partitioned as below.

- Address 'h0 ~ 'h7

  This area is reserved for initialization. The program of FPP0 will be executed from address 'h0 after booting-up; the program of FPP1 will start from address 'h1 after enabled; the program of FPP2 will start from address 'h2 after enabled; the program of FPP3 will start from address 'h3 after enabled, and so on.
- Address 'h8 ~ 'hF

  This area is reserved.
- Address 'h10

  This address is the entry of interrupt service routine.
- Address 'h11 ~ 'h7F7

  This area is for user program.
- Address 'h7F8 ~ 'h7FF

  These addresses are reserved for system use.

Stack Pointer

The stack pointer in each processing unit is used to point the top of the stack area where the local variables and parameters to subroutines are stored; the stack pointer register (sp) is located in IO address 0x02h. The bit number of stack pointer is 8 bit; the stack memory cannot be accessed over 256 bytes and should be defined within 256 bytes from 0x00h address. If the stack is a "full" stack, the stack pointer points to the most recently pushed item, else if it is an "empty" stack, the stack pointer points to the first empty location, where the next item will be pushed. The stack pointer of PDK82C13 for each FPP unit can be assigned by programmer, means that the depth of stack pointer for each FPP unit is adjustable in order to optimize system performance.

Arithmetic and Logic Unit

Arithmetic and Logic Unit (ALU) is the computation element to operate integer arithmetic, logic, shift and other specialized operations. The operation data can be from instruction, accumulator or SRAM data memory. Computation result could be written into accumulator or SRAM.

Program Sequencer

Program Sequencer is a mechanism to decide the program flow that program counter should be filled the next instruction address, filled the branch address or accept interrupt.

16-bit Timer

A 16-bit timer is implemented in the PDK82C13, clock source to the timer16 may come from external crystal clock, internal high RC clock, internal low RC clock or bit 0 of Port A, a multiplex is used to select clock output for the clock source. Before sending clock to the counter16, a pre-scaling logic with divided-by-1, 4, 16, and 64 is used for wide range counting.

The 16-bit counter performs up-counting operation only, the counter initial values can be stored from memory by *stt16* instruction and the counting values can be loaded to memory by *ldt16* instruction. A selector is used to select the interrupt condition of timer16, whenever overflow occurs, the timer16 interrupt can be triggered. The hardware diagram of timer16 is shown as Figure 1. The programmer must be special aware about the clock source to the timer16 when using the ICE system: (1) If system clock is selected as the clock of timer16, the clock to timer16 is also stopped in ICE trap mode (2) If other sources are selected, the clock to timer16 is free running at all time.

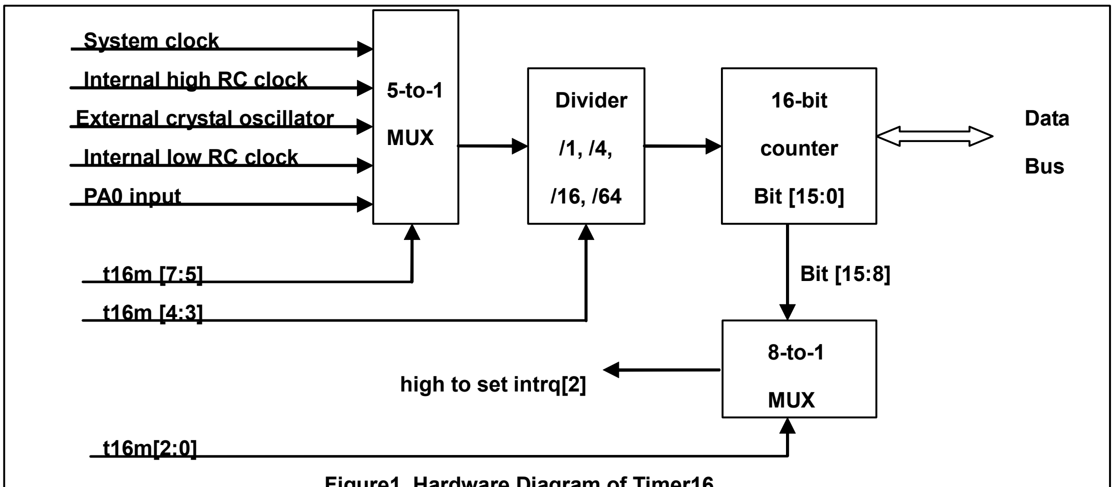


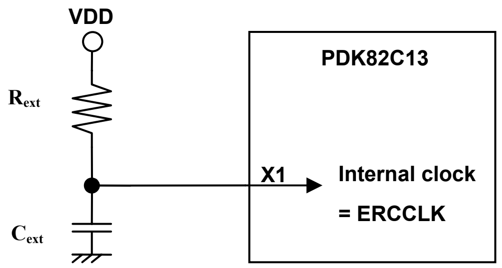
Figure1. Hardware Diagram of Timer16

Oscillator and clock

There are four oscillator circuits in the PDK82C13: external RC oscillator (clock ERCCLK), crystal oscillator (clock EXTALCLK), internal 16MHz high RC oscillator (clock IHRCCLK) and internal low RC oscillator (clock ILRCCLK). In additional to clocks of above four oscillator modules, the system clock can come from external clock source, too. Other than internal low RC oscillator, all the clocks can be divided by 1, 2, and 4 as options to be system clock, and internal low RC oscillator can be divided by 1 or 4, these options can be selected by register clkmd (0x03).

External RC Oscillator

If an external RC oscillator is selected, external resistor and capacitor are needed to generate the required operating frequency. Figure 2 shows the hardware connection of the PDK82C13 and leave X2 as no connection or other application. The bit 3 of register clkmd (0x03) must be set to high before using this oscillator. To consider the stability and noise sensitive, R_{ext} is recommended between 3kΩ and 100kΩ, C_{ext} is recommended between 20pF and 50pF. Although RC oscillator provides cost-effective solution to generate system clock, however, the frequency may drift a lot due to variation of voltage, temperature and process. If accurate timing is required for your application, both external and internal RC oscillators are not suitable.



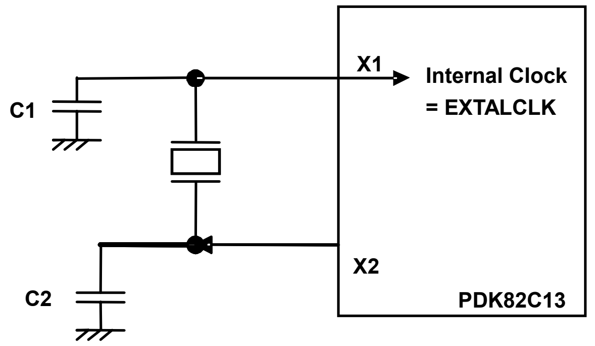
Reference R_{ext} and C_{ext} :

Frequency	R_{ext}	C_{ext}
5MHz	4.7K	22p
600KHz	47K	22p

Figure2. External RC oscillator

Crystal Oscillator

If crystal oscillator is used, a crystal or resonator is required between X1 and X2. Figure 3 shows the hardware connection under this application; the operating frequency of crystal oscillator can range from 32KHz to 16MHz, depending on the crystal placed on. Besides crystal, external capacitor and options of PDK82C13 should be fine tuned in register eoscr (0x0b) to have good sinusoidal waveform.



Reference Frequency	C1 and C2 C1	C2
32KHz	22p	22p
4MHz	22p	22p
16MHz	10p	10p

Figure3. Crystal Oscillator

External Clock Source

If external crystal oscillator circuit or external oscillator device is used to provide clock source to the PDK82C13, its hardware connection is shown as Figure 4, X1 should be leaved as no connection, bit [7:5] of register *eoscr* should be set to 3'b110 and external clock source should be set by bit [7:5] of register *clkmd*. Other than internal low RC oscillator, all the external oscillator circuits will be disabled when PDK82C13 enters the power-down mode in order to reduce power consumption and will be resumed whenever wakeup event is detected; When entering the power-down mode, the internal low RC oscillator may or may not be disabled, depending on the setting of bit 2 of register *clkmd* (0x03). The clock source to the watch-dog timer comes from internal low RC oscillator directly, so the internal low RC oscillator circuit should not be disabled during power-down mode if watch-dog timer is used to wake up system.

(1) eoscr [7:5] = 3'b110 to choose 4M crystal osc
(2) clkmd [7:5] choose external clock
(3) Leave X1 as no connection

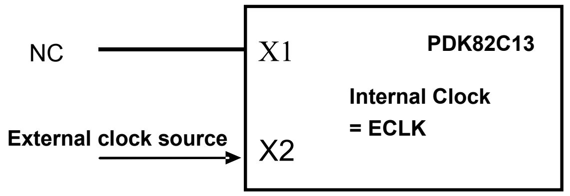


Figure 4. External Clock Source

Watchdog Timer

The watchdog timer is an 11-bit counter with clock coming from internal low RC clock (ILRCCLK), Figure 5 shows its hardware diagram. The frequency of ILRCLK is around 32KHz and the period of watchdog timer is 1024 ILRCCLK, so the time-out period of WDT is around 30ms. WDT can be cleared by power-on-reset or by command *wdreset* at any time. When WDT is timeout, PDK82C13 will be reset to restart the program execution.

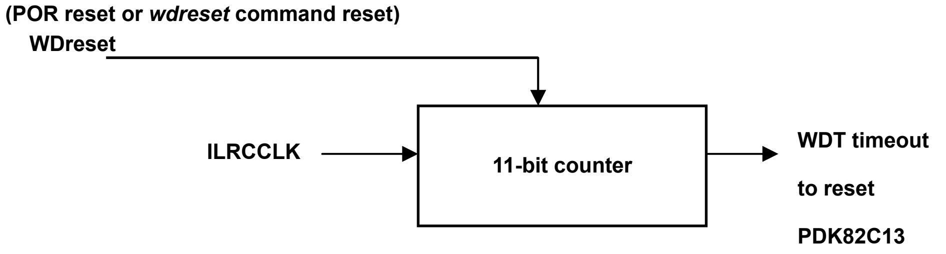


Figure 5. Hardware Diagram of Watchdog Timer

Interrupt

There are eight interrupt lines for PDK82C13: two external interrupt lines (PA0, PB0), Timer16 interrupt and five internal interrupt lines, each interrupt request line has its own corresponding interrupt control bit to enable or disable it. For external interrupt and timer 16, the interrupt request flags are set by hardware and must be cleared by software. For five internal request lines, the interrupt request flags are set by software and cleared by software, too. All the interrupt request lines are also controlled by *engint* command (enable global interrupt) to enable interrupt operation and *disgint* command (disable global interrupt) to disable it. Whenever PDK82C13 jumps to the interrupt address, global interrupt is disabled automatically and enabled automatically whenever *reti* instruction is executed. Interrupt request can be accepted at any time including interrupt service routine period, and the level of interrupt nesting is defined by software because the 8-bit stack pointer register of each FPP unit can be read and written. By adjusting the memory location of stack point, the depth of stack pointer for every FPP unit could be fully specified by user to achieve maximum flexibility of system. The entry address of interrupt service routine is 0x010 no matter the interrupt service routine belongs to which FPP unit.

Power Saving

In order to save power consumption, ON and Power-Down modes are defined by hardware. ON mode is the state of normal operation with all functions ON, Power-Down mode is the state of deeply power-saving with turning off all the high frequency oscillators and leaving internal low frequency RC oscillator for watchdog timer using by option. By using the "*stopsys*" instruction, this chip will be put on Power-Down mode directly. The internal low frequency RC oscillator must be enabled to wakeup the system when in Power-Down mode, means that bit 1 of register *clkmd* (0x03) must be set to high before issuing "*stopsys*" command in order to leave internal low frequency RC oscillator active. The following shows the internal status of PDK82C13 in detail when "*stopsys*" command is issued:

- Both external crystal oscillator and internal high RC oscillator will be turned off.
- Enable internal low RC oscillator (set bit 2 of register clkmd)
- OTP memory is turned off
- The contents of SRAM remain unchanged; however SRAM will be put on Power-Down mode.
- The contents of registers remain unchanged.
- POR circuit is turned off and LVD circuit is active to detect any power glitch.

Besides hardware-defined power-saving states, user can define different power-saving modes by changing the operating frequency in register *clkmd* (0x03). The PDK82C13 can leave the power-down mode by means of (1) external hardware reset (2) LVD detects VDD glitch (3) signals toggle on any input. Wake-up from an external hardware reset or LVD detects VDD glitch will cause PDK82C13 initialization; Wake-up from input pins can be considered as a continuation of normal execution. To minimize power consumption, all the I/O pins should be carefully manipulated before entering power-down mode.

IO Pins

All the bi-directional input/output lines in the PDK82C13 can be configured as different function independently by data registers (*pa, pb*), control registers (*pac, pbc*), pull-high registers (*paph, pbph*) and open-drain registers (*paod*), all these pins have Schmitt-trigger input buffer and output driver with CMOS level. As an example, table 1 shows the configuration table of bit 0 of port A; all other IO lines have the same structure. All the IO pins can be used to wakeup system when PDK82C13 was put in power-down mode.

pa.0	*pac.0*	*paph.0*	*paod.0*	Description
X	0	0	X	Input without pull-up resistor
X	0	1	X	Input with pull-up resistor
0	1	0	0	Output low without pull-up resistor
1	1	0	0	Output high without pull-up resistor
0	1	1	0	Output low with pull-up resistor
1	1	1	0	Output high with pull-up resistor
0	1	0	1	Open drain output low without pull-up resistor
1	1	0	1	Open drain output tri-state without pull-up resistor
0	1	1	1	Open drain output low with pull-up resistor
1	1	1	1	Open drain output tri-state with pull-up resistor

Table1

Reset

There are many conditions to reset the PDK82C13, including:

(1) Power-On Reset (POR)

(2) PRST# pin active in normal operation

(3) PRST# pin active in Power-Down state

(4) WDT timeout in normal operation

(5) WDT timeout in Power-Down state

(6) VDD glitch is detected by LVD

POR (Power-On-Reset) is active to put PDK82C13 in initial state when power-up, watchdog timeout is the abnormal case of software execution, and LVD is used to detect VDD glitch for the abnormal case of power supply. Once reset is asserted, most of all the registers in PDK82C13 will be set to default values, however, some registers keep its content unchanged; System should be restarted once abnormal cases happen, or by jumping program counter to address ’h0. The data memory is in uncertain state when reset comes from power-up and LVD; however, the content will be kept when reset comes from PRST# pin or WDT timeout.

Power-On-Reset (POR)

A power-on reset circuit is built in the PDK82C13, it is used to generate hardware reset signal internally to reset the whole system when power-up of VDD, the POR reset time is longer than 1us to guarantee reset operation for most power-up conditions. Just tie PA5/PRST# to VDD to use this function.

Low-Voltage-Detector (LVD)

The PDK82C13 contains a Low-Voltage-Detector (LVD) circuit that is used to detect the supply voltage spikes during normal operation. Once detecting the low voltage condition, LVD circuit will put the chip into reset state.

Analog-to-Digital Conversion (ADC) module

There are eight input channels for the analog-to-digital conversion module; it allows conversion of an analog input signal to a corresponding 9 ` 10 ` 11 or 12-bit digital number, depending on what the bit resolution is chosen. The hardware block diagram of ADC module is shown as Fig.7; the output of the sample and hold is the input into the converter which generates the result via successive approximation. The analog reference high voltage is software selectable to either the device's analog positive supply voltage (AVDD) or the voltage level on the PB1 pin; the analog reference low voltage is also software selectable to either the device's analog negative supply voltage (AGND) or the voltage level on the PB2 pin.

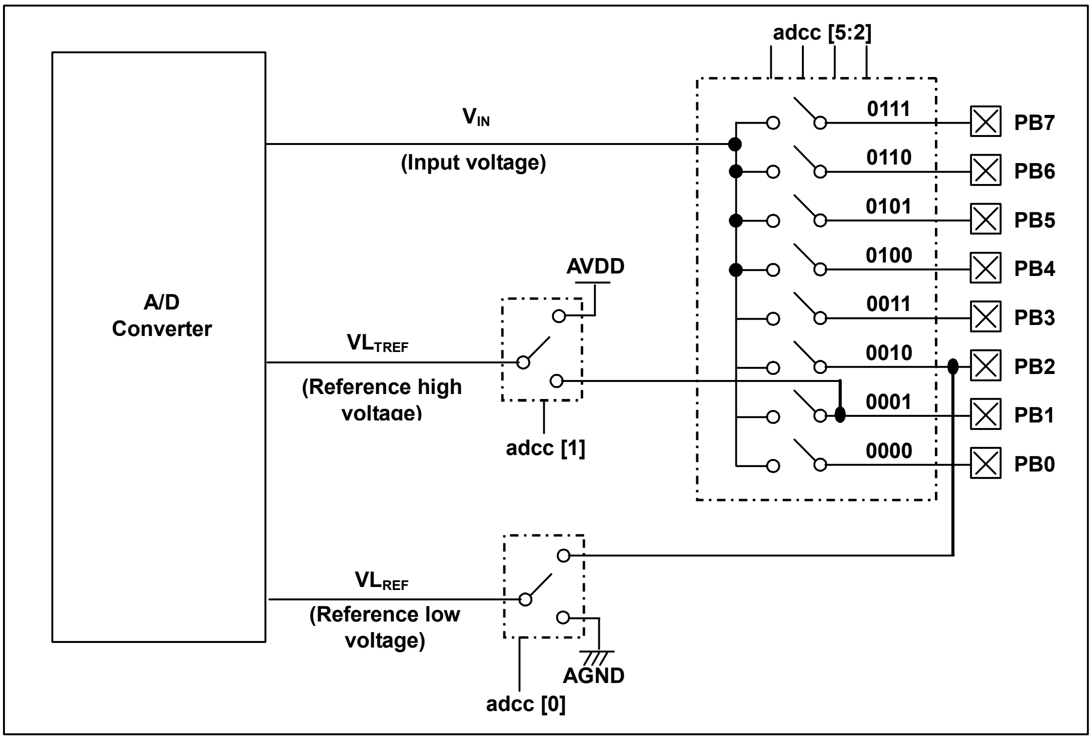


Figure7. ADC Block Diagram

There are five registers for the ADC module, which are:

- ADC Control Register (*adcc*)
- ADC Mode Register (*adcm*)
- ADC Result High Register (*adcrh*)
- ADC Result Low Register (*adcrl*)
- Analog Input Control Register (*aic*)

A device reset will force all registers to their reset state, the ADC module will be turned off, and any conversion will be aborted. The device reset may come from hardware reset, power-on reset, LVD reset, watchdog timeout reset and system error reset. The *adcc* register controls the operation of the ADC module, the *adcm* register defines the resolution and operating clock of the ADC function, the pin of port B can be configured as analog inputs or as digital IO via the *aic* register. When the AD conversion is complete, the high byte result is latched into the *adcrh* register and the low byte result is latched into the *adcrl* register.

After the ADC module has been configured as desired and the selected channel has been configured as analog input. The selected signal should be acquired before conversion, and the AD conversion can be started after the acquisition time has elapsed. The following steps should be followed to do the AD conversion:

1. Configure the ADC module:
   - Configure the voltage reference high and voltage reference low by *adcc* register
   - Select the ADC input channel by *adcc* register
   - Select the bit resolution of ADC by *adcm* register
   - Configure the AD conversion clock by *adcm* register
   - Configure the selected pin as analog input by *aic* register
   - Enable the ADC module by *adcc* register
2. Configure interrupt for ADC: (if desired)
   - Clear the ADC interrupt request flag in bit 3 of *intrq* register
   - Enable the ADC interrupt request in bit 3 of *inten* register
   - Enable global interrupt by issuing *engint* command
3. Start AD conversion:
   - Set ADC process control bit in the *adcc* register to start the conversion
4. Wait for the completion flag of AD conversion, by either:
   - Waiting for the completion flag by using command "*wait1 addc*.6"; or
   - Waiting for the ADC interrupt.
5. Read the ADC result registers:
   - Read *adcrh* and *adcrl* the result registers
6. For next conversion, goto step 1 or step 2 as required.

The input requirement for AD conversion

For the AD conversion to meet its specified accuracy, the charge holding capacitor (C_{HOLD}) must be allowed to fully charge to the voltage reference high level and discharge to the voltage reference low level. The analog input model is shown as Fig.8, the signal driving source impedance (Rs) and the internal sampling switch impedance (Rss) will affect the required time to charge the capacitor C_{HOLD} directly. The internal sampling switch impedance may vary with ADC supply voltage (AVDD), the signal driving source impedance will affect the offset voltage at the analog input due to pin leakage current. The recommended maximum impedance for analog driving source is 10KΩ.

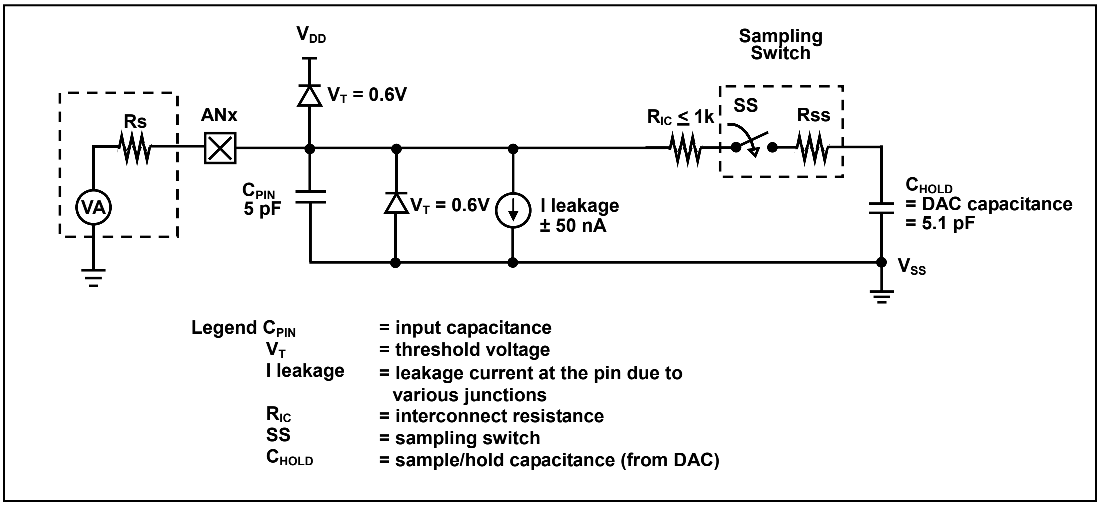


Figure8. Analog Input Model

Before starting the AD conversion, the minimum signal acquisition time should be met for the selected analog input signal. The signal acquisition time (T_{ACQ}) of ADC in PDK82C13 series is fixed to one clock period of ADCLK, the selection of ADCLK must be met the minimum signal acquisition time.

Selecting the ADC bit resolution

The ADC bit resolution is also selectable from 8-bit to 12-bit, depending on the requirement of customers' application. Higher resolution can detect small signal variation; however, it will take more time to convert the analog signal to digital signal. The selection can be done via *adcm* register. The ADC bit resolution should be configured before starting the AD conversion.

ADC clock selection

The clock of ADC module (ADCLK) can be selected by *adcm* register; there are 12 possible options for ADCLK from sysclk/1 to sysclk/2048. Due to the signal acquisition time T_{ACQ} is one clock period of ADCLK, the ADCLK must meet that requirement.

AD conversion

The process of AD conversion starts from setting START/DONE bit (bit 6 of *adcc*) to high, the START/DONE flag for read will be cleared automatically, then converting analog signal bit by bit and finally setting START/DONE high to indicate the completion of AD conversion. If ADCLK is selected, T_{ADCLK} is the period of ADCLK and the AD conversion time can be calculated as follows:

- **8-bit resolution: AD conversion time = 12 T_{ADCLK}**
- **9-bit resolution: AD conversion time = 13 T_{ADCLK}**
- **10-bit resolution: AD conversion time = 14 T_{ADCLK}**
- **11-bit resolution: AD conversion time = 15 T_{ADCLK}**
- **12-bit resolution: AD conversion time = 16 T_{ADCLK}**

Configuring the analog pins

The eight analog input signals for ADC share the same pins with port B and the default setting is for digital signal. To avoid leakage current at the digital circuit, those pins defined for analog input should be set to be analog input via *aic* register. For those defined analog input pins, the value will be 0 when reading port B.

IO Registers Address and Description

The address mapping of IO registers is the following:

Name	Address	Function	POR/LVD reset	PRST# or WDT reset
flag	0x00	Arithmetic status flag	vvvv_0000	vvvv_0000
fppen	0x01	FPP unit enable register	0000_0001	0000_0001
sp	0x02	Stack pointer	xxxx_xxxx	uuuu_uuuu
clkmd	0x03	Clock mode register	1111_0110	1111_0110
inten	0x04	Interrupt enable register	xxxx_xxxx	uuuu_uuuu
intrq	0x05	Interrupt request register	xxxx_xxxx	uuuu_uuuu
t16m	0x06	Timer 16 mode register	0000_0000	0000_0000
gdio	0x07	General data register for IO	0000_0000	uuuu_uuuu
	0x08 ~ 0x09	Reserved		
eoscr	0x0a	External oscillator setting register	0000_0000	0000_0000
ihrcr	0x0b	Internal high RC oscillator control register	0000_0000	0000_0000
	0x0c ~ 0x0f	Reserved		
pa	0x10	Port A data register	0000_0000	0000_0000
pac	0x11	Port A control register	0000_0000	0000_0000
paph	0x12	Port A pull high register	0000_0000	0000_0000
paod	0x13	Port A open drain register	0000_0000	0000_0000
pb	0x14	Port B data register	0000_0000	0000_0000
pbc	0x15	Port B control register	0000_0000	0000_0000
pbph	0x16	Port B pull high register	0000_0000	0000_0000
adcc	0x20	AD control register	0000_0000	0000_0000
adcm	0x21	AD mode register	0000_0000	0000_0000
adcrh	0x22	AD result high register	xxxx_xxxx	xxxx_xxxx
adcrl	0x23	AD result low register	xxxx_xxxx	xxxx_xxxx
adcdi	0x24	AD analog input disable register	0000_0000	0000_0000
U: unchanged, v: reserved, x: unknown				

ACC Status Flag Register (*flag*), IO address = 0x00

Bit	Reset	R/W	Description
7 - 4	-	-	Reserved. These four bits are "1" when read.
3	0	R/W	OV (Overflow). This bit is set whenever the sign operation is overflow.
2	0	R/W	AC (Auxiliary Carry). There are two conditions to set this bit, the first one is carry out of low nibble in addition operation, and the other one is no borrow from the high nibble into low nibble in subtraction operation.
1	0	R/W	C (Carry). There are two conditions to set this bit, the first one is carry out in addition operation, and the other one is no borrow in subtraction operation. Carry is also affected by shift with carry instruction.
0	0	R/W	Z (Zero). This bit will be set when the result of arithmetic or logic operation is zero; Otherwise, it is cleared.

FPP unit Enable Register (*fppen*), IO address = 0x01

Bit	Reset	R/W	Description
7	0	R/W	FPP7 enable. This bit is used to enable FPP7. 0 / 1: disable / enable
6	0	R/W	FPP6 enable. This bit is used to enable FPP6. 0 / 1: disable / enable
5	0	R/W	FPP5 enable. This bit is used to enable FPP5. 0 / 1: disable / enable
4	0	R/W	FPP4 enable. This bit is used to enable FPP4. 0 / 1: disable / enable
3	0	R/W	FPP3 enable. This bit is used to enable FPP3. 0 / 1: disable / enable
2	0	R/W	FPP2 enable. This bit is used to enable FPP2. 0 / 1: disable / enable
1	0	R/W	FPP1 enable. This bit is used to enable FPP1. 0 / 1: disable / enable
0	1	R/W	FPP0 enable. This bit is used to enable FPP0. 0 / 1: disable / enable

Stack Pointer Register (*sp*), IO address = 0x02

Bit	Reset	R/W	Description
7 - 0	-	R/W	Stack Pointer Register. Read out the current stack pointer, or write to change the stack pointer.

Clock Mode Register (*clkmd*), IO address = 0x03

Bit	Reset	R/W	Description
7 – 5	111	R/W	System clock selection 000: internal high RC/4 001: internal high RC/2 010: internal high RC 011: external OSC/4 100: external OSC/2 101: external OSC 110: internal low RC/4 111: internal low RC Note: external OSC: external RC, crystal oscillators and external clock input
4	1	R/W	Internal High RC Enable. 0 / 1: disable / enable
3	0	-	Reserved. Must be "0".
2	1	R/W	Internal Low RC Enable. 0 / 1: disable / enable

1	1	R/W	Watch Dog Enable. 0 / 1: disable / enable
0	0	R/W	Pin PA5/PRST# function. 0 / 1: PA5 / PRST#.

Interrupt Enable Register (*inten*), IO address = 0x04

Bit	Reset	R/W	Description
7	-	R/W	Enable general interrupt bit 4. 0 / 1: disable / enable.
6	-	R/W	Enable general interrupt bit 3. 0 / 1: disable / enable.
5	-	R/W	Enable general interrupt bit 2. 0 / 1: disable / enable.
4	-	R/W	Enable general interrupt bit 1. 0 / 1: disable / enable.
3	-	R/W	Enable interrupt from ADC. 0 / 1: disable / enable.
2	-	R/W	Enable interrupt from timer16 overflow. 0 / 1: disable / enable.
1	-	R/W	Enable interrupt from pb0. 0 / 1: disable / enable.
0	-	R/W	Enable interrupt from pa0.0 / 1: disable / enable.

Interrupt Request Register (*intrq*), IO address = 0x05

Bit	Reset	R/W	Description
7	-	R/W	Interrupt Request 4, this bit is set and clear by software. 0 / 1 : No request / Request
6	-	R/W	Interrupt Request 3, this bit is set and clear by software. 0 / 1 : No request / Request
5	-	R/W	Interrupt Request 2, this bit is set and clear by software. 0 / 1 : No request / Request
4	-	R/W	Interrupt Request 1, this bit is set and clear by software. 0 / 1 : No request / Request
3	-	R/W	Interrupt Request from ADC, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
2	-	R/W	Interrupt Request from timer16, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
1	-	R/W	Interrupt Request from pin PB0, this bit is set by hardware and cleared by software. 0 / 1: No request / Request
0	-	R/W	Interrupt Request from pin PA0, this bit is set by hardware and cleared by software. 0 / 1: Request / No request

Timer 16 mode Register (*t16m*), IO address = 0x06

Bit	Reset	R/W	Description
7 - 5	000	R/W	Timer Clock source selection. 001: system clock 100: internal high RC 101: external OSC 110: internal low RC 111: PA.0 (external event) Others: Timer 16 is disabled
4 – 3	00	R/W	Internal clock divider. 00: /1 01: /4 10: /16 11: /64

2 – 0	000	R/W	Interrupt source selection. Interrupt event happens when selected bit goes high. 0 : bit 8 of timer16 1 : bit 9 of timer16 2 : bit 10 of timer16 3 : bit 11 of timer16 4 : bit 12 of timer16 5 : bit 13 of timer16 6 : bit 14 of timer16 7 : bit 15 of timer16
Application Note: In order to have accurate counting result, the 16-bit counter should be initialized after writing this register. The programmer must be special aware about the clock source to the timer16 during the ICE system: (1) If system clock is chosen as the clock of timer16, the clock to timer16 is also stopped in ICE trap mode (2) If other sources are chosen, the clock to timer16 is free running at all time.			

General Data register for IO (*gdio*), IO address = 0x07

Bit	Reset	R/W	Description
7 – 0	00	R/W	General data for IO. This port is the general data buffer in IO space and cleared only when POR, LVD or pin PRST# is active. It can perform the IO operation, like ***wait0*** gdio.x, ***wait1*** gdio.x and ***tog*** gdio.x to take the replace of operations which instructions are supported in memory space (ex: ***wait1*** mem; ***wait0*** mem; ***tog*** mem).

External Oscillator setting Register (*eoscr*), IO address = 0x0a

Bit	Reset	R/W	Description
7	0	R/W	Enable external RC oscillator or crystal oscillator. 0 / 1 : Disable / Enable
6 – 5	00	R/W	External oscillator selection. 00 : external RC oscillator 01 : 32KHz crystal oscillator 10 : 4MHz crystal oscillator 11 : 16MHz crystal oscillator
4 – 0	10000	R/W	Options for external crystal oscillator; please see the application note.

Internal High RC oscillator control Register low (*ihrcr*), IO address = 0x0b

Bit	Reset	R/W	Description
7 – 0	00	R/W	Bit [7:0] of internal high RC oscillator for speed calibration.

Port A, B Data Registers (*pa, pb*), IO address = 0x10, 0x14

Bit	Reset	R/W	Description
7 – 0	8'h00	R/W	Data registers for Port A, B.

Port A, B Control Registers (*pac, pbc*), IO address = 0x11, 0x15

Bit	Reset	R/W	Description
7 - 0	8'h00	R/W	Port A, B control registers. These registers are used to define input mode or output mode for each corresponding pin of port A, B. 0 / 1: input / output <u>Please note that the bit 5 of port A (PA5) is input only</u>.

Port A, B Pull-High Registers (*paph, pbph*), IO address = 0x12, 0x16

Bit	Reset	R/W	Description
7 - 0	8'h00	R/W	Port A, B pull-high registers. These registers are used to enable the internal pull-high device on each corresponding pin of port A, B. 0 / 1 : disable / enable Please note that the bit 5 of port A (PA5) does not have pull-up resistor.

Port A Open-Drain Registers (*paod*), IO address = 0x13

Bit	Reset	R/W	Description
7 – 0	8'h00	R/W	Port A open-drain registers. This register is used to set the output buffer configuration on each corresponding pin of port A. 0 / 1 : Hi-Lo two states output / Open-drain output Please note that the bit 5 of port A (PA5) is input only.

ADC Control Register (*adcc*), IO address = 0x20

Bit	Reset	R/W	Description
7	0	R/W	Enable ADC function. 0/1: Disable/Enable.
6	0	R/W	ADC process control bit. Write "1" to start AD conversion, and the completion flag is cleared automatically; Read "1" to indicate the completion of AD conversion.
5 – 2	0000	R/W	Channel selector. These three bits are used to select input signal for AD conversion. 1XXX:Reserved 0000:PB0, 0001:PB1, 0010:PB2, 0011:PB3, 0100:PB4, 0101:PB5, 0110:PB6, 0111:PB7
1	0	R/W	Vref high selector. This bit is used to select the source as Vref high. 0/1: AVDD/PB1
0	0	R/W	Vref low selector. This bit is used to select the source as Vref low. 0/1: AGND/PB2

ADC Mode Register (*adcm*), IO address = 0x21

Bit	Reset	R/W	Description
7 – 5	000	R/W	Bit Resolution. 000:8-bit, 001:9-bit, 010:10-bit, 011:11-bit, 100:12-bit, others: reserved
4 – 1	0000	R/W	ADC clock source selection. 0000:sysclk/1, 0001:sysclk/2, 0010:sysclk/4, 0011:sysclk/8, 0100:sysclk/16, 0101:sysclk/32, 0110:sysclk/64, 0111:sysclk/128, Others: reserved.
0	-	-	Reserved

ADC Result High Register (*adcrh*), IO address = 0x22

Bit	Reset	R/W	Description
7 – 0	-	R/O	These eight read-only bits will be the bit [11:4] of AD conversion result.

ADC Result Low Register (*adcrl*), IO address = 0x23

Bit	Reset	R/W	Description
7 – 4	-	R/O	These four bits will be the bit [3:0] of AD conversion result.
3 – 0	-	-	Reserved

Analog Input Control Register (*adcdi*), IO address = 0x24

Bit	Reset	R/W	Description
7	0	R/W	PB7 input: 0/1: digital input/analog input.
6	0	R/W	PB6 input: 0/1: digital input/analog input.
5	0	R/W	PB5 input: 0/1: digital input/analog input.
4	0	R/W	PB4 input: 0/1: digital input/analog input.
3	0	R/W	PB3 input: 0/1: digital input/analog input.
2	0	R/W	PB2 input: 0/1: digital input/analog input.
1	0	R/W	PB1 input: 0/1: digital input/analog input.
0	0	R/W	PB0 input: 0/1: digital input/analog input.

Instructions

Symbol	Description
ACC	Accumulator
a	Accumulator
sp	Stack pointer
flag	ACC status flag register
I	Immediate data
&	Logical AND
|	Logical OR
←	Movement
^	Exclusive logic OR
+	Add
－	Subtraction
～	NOT (logical complement, 1's complement)
〒	NEG (2's complement)
OV	Overflow (The operational result is out of range in signed 2's complement number system)
Z	Zero (If the result of ALU operation is zero, this bit is set to 1)
C	Carry (The operational result is to have carry out for addition or to borrow carry for subtraction in unsigned number system)
AC	Auxiliary Carry (If there is a carry out from low nibble after the result of ALU operation, this bit is set to 1)
pc0	Program counter for FPP0
pc1	Program counter for FPP1
pc2	Program counter for FPP2
pc3	Program counter for FPP3
pc4	Program counter for FPP4
pc5	Program counter for FPP5
pc6	Program counter for FPP6
pc7	Program counter for FPP7

Data Transfer Instructions (20)

mov a, I	Move immediate data into ACC. Example: *mov* a, 0x0f; Result: a ← 0fh; Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
mov M, a	Move data from ACC into memory Example: *mov* MEM, a; Result: MEM ← a Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
mov a, M	Move data from memory into ACC Example: *mov* a, MEM ; Result: a ← MEM; Flag Z is set when MEM is zero. Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
mov a, IO	Move data from IO into ACC Example: *mov* a, pa ; Result: a ← pa; Flag Z is set when pa is zero. Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV
mov IO, a	Move data from ACC into IO Example: *mov* pb, a; Result: pb ← a Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV
nmov M, a	Take the negative logic (2's complement) of ACC to put on memory Example: *mov* MEM, a; Result: MEM ← 〒a Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV Application Example: -- *mov* a, 0xf5 ; // ACC is 0xf5 *nmov* ram9, a; // ram9 is 0x0b, ACC is 0xf5 --
nmov a, M	Take the negative logic (2's complement) of memory to put on ACC Example: *mov* a, MEM ; Result: a ← 〒MEM; Flag Z is set when 〒MEM is zero. Affected flags: 『Y』 Z 『N』 C 『N』 AC 『N』 OV Application Example: -- *mov* a, 0xf5 ; *mov* ram9, a ; // ram9 is 0xf5 *nmov* a, ram9 ; // ram9 is 0xf5, ACC is 0x0b --
pushw *word*	Move the source data from the memory in *word* to memory that address specified in the stack pointer (*pushw word*). It needs 2T to execute this instruction. Example: *pushw* *word*; Result: [sp] ← *word* ; sp ← sp + 2 ; Affected flags: 『N』 Z 『N』 C 『N』 AC 『N』 OV

	Application Example: -- word ptr0 ; // declare pointer in RAM … *mov* a, 0x55 ; *mov* ld@ptr0, a ; // move 0x55 to RAM with ptr0 pointer (LSB) *mov* a, 0xaa ; *mov* hd@ptr0, a ; // move 0xaa to RAM with ptr0 pointer (MSB) *pushw* ptr0 ; // move (0xaa, 0x55) to stack memory … --
pushw pcN	Store the program counter of Nth FPP unit to the memory which address is specified in the stack pointer of current executing FPP unit (*pushw* pcN). It needs 2T to execute this instruction. Please notice that the target FPP unit should be disabled before issuing this command. Example: *pushw* pc3; (Instruction is executed by FPP0 as this example) Result: [sp] of FPP0 ← pc of FPP3 ; sp of FPP0 ← sp of FPP0 + 2 ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: -- fpp0_loop: … *set0* fppen.1 ; // disable FPP1 by FPP0, PC1 (PC of FPP1) @ 0x0123 *pushw* pc1 ; // store PC1(0x0123) to stack memory … *goto* fpp0_loop ; fpp1_loop: … … // When disabled by FPP0, PC of FPP1=0x0123 … *goto* fpp1_loop ; --
popw word	Move the memory data from the address specified in the stack pointer to the *word* memory (*popw word*). It needs 2T to execute this instruction. Example: *popw word*; Result: sp ← sp - 2; word ← [sp] ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: -- word ptr0 ; // declare 1st pointer in RAM word ptr1 ; // declare 2nd pointer in RAM … *mov* a, 0x55 ;

	mov ld@ptr0, a ; // move 0x55 to RAM with ptr0 pointer (LSB) *mov* a, 0xaa ; *mov* hd@ptr0, a ; // move 0xaa to RAM with ptr0 pointer (MSB) *pushw* ptr0 ; // move (0xaa, 0x55) to stack memory (word) *popw* ptr1 ; // move (0xaa, 0x55) to RAM with ptr1 pointer (word) *mov* a, ld@ptr1 ; // ACC=0x55 *mov* a, hd@ptr1 ; // ACC=0xaa --
popw pcN	Restore the program counter of the Nth FPP unit from the memory which address is specified in the stack pointer of current executing FPP unit (*popw* pcN). It needs 2T to execute this instruction. Please notice that the target FPP unit should be disabled before issuing this command. Example: *popw* pc3; (Instruction is executed by FPP0 as this example) Result: sp of FPP0 ← sp of FPP0 - 2 ; pc of FPP3 ← [sp] of FPP0 ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example 1 : -- fpp0_loop: … *set0* fppen.1 ; // freeze PC1(0x0123) by disabling FPP1 from FPP0 *pushw* pc1 ; // store PC1(0x0123) to stack memory specified by FPP0 *nop;* … *set0* fppen.1 ; // disable FPP unit before restoring PC *popw* pc1 ; // restore PC1 from stack memory of FPP0 *set1* fppen.1 ; // free FPP1 to run program continuous … *goto* fpp0_loop ; fpp1_loop: … … // When disabled by FPP0, PC of FPP1=0x0123 … *goto* fpp1_loop ; -- Application Example 2 : -- word ptr0 ; // declare a RAM pointer … fpp0_loop: … *mov* a, la@Codelabel ; // move a label to pointer (LSB) *mov* lb@ptr0, a ; *mov* a, ha@Codelabel ; // move a label to pointer (MSB) *mov* hb@ptr0, a ; *pushw* ptr0 ; // push the Codelabel address to stack memory *popw* pc5 ; // pop the stack content to be the PC of FPP5 // request FPP5 to jump to "Codelabel" immediately … Codelabel: … --
ldtabh index	Load high byte data in OTP program memory to ACC by using index as OTP address. It needs 2T to execute this instruction. Example: *ldtabh* index;

	Result: a ← {bit 15~8 of OTP [index]}; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: -- word ROMptr ; // declare a pointer of ROM in RAM … *mov* a, la@TableA ; // assign pointer to ROM TableA (LSB) *mov* lb@ROMptr, a ; // save pointer to RAM (LSB) *mov* a, ha@TableA ; // assign pointer to ROM TableA (MSB) *mov* hb@ROMptr, a ; // save pointer to RAM (MSB) … *ldtabh* ROMptr ; // load TableA MSB to ACC (ACC=0X02) …. TableA : dc 0x0234, 0x0042, 0x0024, 0x0018 ; --
ldtabl index	Load low byte data in OTP to ACC by using index as OTP address. It needs 2T to execute this instruction. Example: *ldtabl* index; Result: a ← {bit7~0 of OTP [index]}; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: -- word ROMptr ; // declare a pointer of ROM in RAM … *mov* a, la@TableA ; // assign pointer to ROM TableA (LSB) *mov* lb@ROMptr, a ; // save pointer to RAM (LSB) *mov* a, ha@TableA ; // assign pointer to ROM TableA (MSB) *mov* hb@ROMptr, a ; // save pointer to RAM (MSB) … *ldtabl* ROMptr ; // load TableA LSB to ACC (ACC=0x34) …. TableA : dc 0x0234, 0x0042, 0x0024, 0x0018 ; --
ldt16 index	Move 16-bit counting values in Timer16 to memory that is addressed by index. Example: *ldt16* index; Result: [index] ← [16-bit timer] Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: -- word T16ptr ; // declare a RAM pointer … *clear* lb@T16ptr ; // clear T16 memory pointer (LSB)

	clear hb@T16ptr ; // clear T16 memory pointer (MSB) *stt16* T16ptr ; // initial T16 with 0 … *set1* t16m.5 ; // enable Timer16 … *set0* t16m.5 ; // disable Timer 16 *ldt16* T16ptr ; // save the T16 counting value to RAM index by T16ptr …. ---
stt16 index	Store 16-bit data from memory addressed by index to Timer16. Example: *stt16* index; Result: [16-bit timer] ← [index] Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: --- word T16ptr ; // declare a RAM pointer … *mov* a, 0x34 ; *mov* lb@T16ptr, a ; // move 0x34 to memory indexed by T16ptr (LSB) *mov* a, 0x12 ; *mov* hb@T16ptr, a ; // move 0x12 to memory indexed by T16ptr (MSB) *stt16* T16ptr ; // initial T16 with 0x1234 … ---
idxm a, index	Move data from specified memory to ACC by indirect method. It needs 2T to execute this instruction. Example: *idxm* a, index; Result: index1 ← [index], a ← [index1], where index and index1 are declared by word. Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV Application Example: --- word RAMIndex ; // declare a RAM pointer … *mov* a, 0x5B ; // assign pointer to an address (LSB) *mov* lb@RAMIndex, a ; // save pointer to RAM (LSB) *mov* a, 0x00 ; // assign 0x00 to an address (MSB), should be 0 *mov* hb@RAMIndex, a ; // save pointer to RAM (MSB) … *idxm* a, RAMIndex ; // mov memory data in address 0x5B to ACC ---
Idxm index, a	Move data from ACC to specified memory by indirect method. It needs 2T to execute this instruction. Example: *idxm* index, a; Result: index1 ← [index], [index1] ← a; where index and index1 are declared by word. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

	Application Example: -- word RAMIndex ; // declare a RAM pointer … *mov* a, 0x5B ; // assign pointer to an address (LSB) *mov* lb@RAMIndex, a ; // save pointer to RAM (LSB) *mov* a, 0x00 ; // assign 0x00 to an address (MSB), should be 0 *mov* hb@RAMIndex, a ; // save pointer to RAM (MSB) … *mov* a, 0xA5 ; *idxm* RAMIndex, a ; // mov 0xA5 to memory in address 0x5B --
xch M	Exchange data between ACC and memory Example: *xch* MEM ; Result: MEM ← a , a ← MEM Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
pushaf	Move the *ACC* and *flag* register to memory that address specified in the stack pointer. Example: *pushaf*; Result: [sp] ← {flag, ACC}; sp ← sp + 2 ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: -- .romadr 0x10 ; // ISR entry address *pushaf* ; // put ACC and flag into stack memory … // ISR program … // ISR program *popaf* ; // restore ACC and flag from stack memory *reti* ; --
popaf	Restore *ACC* and *flag* from the memory which address is specified in the stack pointer. Example: *popaf*; Result: sp ← sp - 2 ; {Flag, ACC} ← [sp] ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

Arithmetic Operation Instructions (19)

add a, I	Add immediate data with ACC, then put result into ACC Example: *add* a, 0x0f ; Result: a ← a + 0fh Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
add a, M	Add data in memory with ACC, then put result into ACC Example: *add* a, MEM ;

	Result: a ← a + MEM Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
add M, a	Add data in memory with ACC, then put result into memory Example: *add* MEM, a; Result: MEM ← a + MEM Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
addc a, M	Add data in memory with ACC and carry bit, then put result into ACC Example: *addc* a, MEM ; Result: a ← a + MEM + C Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
addc M, a	Add data in memory with ACC and carry bit, then put result into memory Example: *addc* MEM, a ; Result: MEM ← a + MEM + C Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
addc a	Add carry with ACC, then put result into ACC Example: *addc* a ; Result: a ← a + C Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
addc M	Add carry with memory, then put result into memory Example: *addc* MEM ; Result: MEM ← MEM + C Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
nadd a, M	Add negative logic (2's complement) of ACC with memory Example: *nadd* a, MEM ; Result: a ← ╤a + MEM Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
nadd M, a	Add negative logic (2's complement) of memory with ACC Example: *nadd* MEM, a ; Result: MEM ← ╤MEM + a Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
sub a, I	Subtraction immediate data from ACC, then put result into ACC. Example: *sub* a, 0x0f; Result: a ← a - 0fh (a + [2's complement of 0fh]) Affected flags: 『Y』 Z 『Y』 C 『Y』 AC 『Y』 OV
sub a, M	Subtraction data in memory from ACC, then put result into ACC Example: *sub* a, MEM ;

	Result: a ← a - MEM (a + [2's complement of M]) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
sub M, a	Subtraction data in ACC from memory, then put result into memory Example: *sub* MEM, a; Result: MEM ← MEM - a (MEM + [2's complement of a]) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc a, M	Subtraction data in memory and carry from ACC, then put result into ACC Example: *subc* a, MEM; Result: a ← a – MEM - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc M, a	Subtraction ACC and carry bit from memory, then put result into memory Example: *subc* MEM, a ; Result: MEM ← MEM – a - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc a	Subtraction carry from ACC, then put result into ACC Example: *subc* a; Result: a ← a - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
subc M	Subtraction carry from the content of memory, then put result into memory Example: *subc* MEM; Result: MEM ← MEM - C Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
inc M	Increment the content of memory Example: *inc* MEM ; Result: MEM ← MEM + 1 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
dec M	Decrement the content of memory Example: *dec* MEM; Result: MEM ← MEM - 1 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
clear M	Clear the content of memory Example: *clear* MEM ; Result: MEM ← 0 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

Shift Operation Instructions (10)

sr a	Shift right of ACC Example: *sr* a ; Result: a (0,b7,b6,b5,b4,b3,b2,b1) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b0) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
src a	Shift right of ACC with carry Example: *src* a ; Result: a (c,b7,b6,b5,b4,b3,b2,b1) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b0) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
sr M	Shift right the content of memory Example: *sr* MEM ; Result: MEM(0,b7,b6,b5,b4,b3,b2,b1) ← MEM(b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b0) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
src M	Shift right of memory with carry Example: *src* MEM ; Result: MEM(c,b7,b6,b5,b4,b3,b2,b1) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b0) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
sl a	Shift left of ACC Example: *sl* a ; Result: a (b6,b5,b4,b3,b2,b1,b0,0) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a (b7) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
slc a	Shift left of ACC with carry Example: *slc* a ; Result: a (b6,b5,b4,b3,b2,b1,b0,c) ← a (b7,b6,b5,b4,b3,b2,b1,b0), C ← a(b7) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
sl M	Shift left of memory Example: *sl* MEM ; Result: MEM (b6,b5,b4,b3,b2,b1,b0,0) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM(b7) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
slc M	Shift left of memory with carry Example: *slc* MEM ; Result: MEM (b6,b5,b4,b3,b2,b1,b0,C) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0), C ← MEM (b7) Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV
swap a	Swap the high nibble and low nibble of ACC Example: *swap* a ; Result: a (b3,b2,b1,b0,b7,b6,b5,b4) ← a (b7,b6,b5,b4,b3,b2,b1,b0) Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
swap M	Swap the high nibble and low nibble of memory

	Example: *swap* MEM ; Result: MEM (b3,b2,b1,b0,b7,b6,b5,b4) ← MEM (b7,b6,b5,b4,b3,b2,b1,b0) Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

Logic Operation Instructions (16)

and a, I	Perform logic AND on ACC and immediate data, then put result into ACC Example: *and* a, 0x0f ; Result: a ← a & 0fh Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
and a, M	Perform logic AND on ACC and memory, then put result into ACC Example: *and* a, RAM10 ; Result: a ← a & RAM10 Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
and M, a	Perform logic AND on ACC and memory, then put result into memory Example: *and* MEM, a ; Result: MEM ← a & MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
or a, I	Perform logic OR on ACC and immediate data, then put result into ACC Example: *or* a, 0x0f ; Result: a ← a | 0fh Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
or a, M	Perform logic OR on ACC and memory, then put result into ACC Example: *or* a, MEM ; Result: a ← a | MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
or M, a	Perform logic OR on ACC and memory, then put result into memory Example: *or* MEM, a ; Result: MEM ← a | MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
xor a, I	Perform logic XOR on ACC and immediate data, then put result into ACC Example: *xor* a, 0x0f ; Result: a ← a ^ 0fh Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
xor a, M	Perform logic XOR on ACC and memory, then put result into ACC Example: *xor* a, MEM ; Result: a ← a ^ RAM10 Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
xor M, a	Perform logic XOR on ACC and memory, then put result into memory

	Example: *xor* MEM, a ; Result: MEM ← a ^ MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV
not a	Perform 1's complement (logical complement) of ACC Example: *not* a ; Result: a ← ～a Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV Application Example: -- *mov* a, 0x38 ; // ACC=0X38 *not* a ; // ACC=0XC7 --
not M	Perform 1's complement (logical complement) of memory Example: *not* MEM ; Result: MEM ← ～MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV Application Example: -- *mov* a, 0x38 ; *mov* mem, a ; // mem = 0x38 *not* mem ; // mem = 0xC7 --
neg a	Perform 2's complement of ACC Example: *neg* a ; Result: a ← 〒a Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV Application Example: -- *mov* a, 0x38 ; // ACC=0X38 *neg* a ; // ACC=0XC8 --
neg M	Perform 2's complement of memory Example: *neg* MEM ; Result: MEM ← 〒MEM Affected flags: 『Y』Z 『N』C 『N』AC 『N』OV Application Example: -- *mov* a, 0x38 ; *mov* mem, a ; // mem = 0x38 *not* mem ; // mem = 0xC8 --
comp a, I	Compare ACC with immediate data

	Example: *comp* a, 0x55; Result: Flag will be changed by regarding as (a - 0x55) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV Application Example: --- *mov* a, 0x38 ; *comp* a, 0x38 ; // Z flag is set *comp* a, 0x42 ; // C flag is set *comp* a, 0x24 ; // C, Z flags are clear *comp* a, 0x6a ; // C, AC, OV flags are set ---
comp a, M	Compare ACC with the content of memory Example: *comp* a, MEM; Result: Flag will be changed by regarding as (a - MEM) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV Application Example: --- *mov* a, 0x38 ; mov mem, a ; comp a, mem ; // Z flag is set mov a, 0x42 ; mov mem, a ; mov a, 0x38 ; comp a, mem ; // C flag is set ---
comp M, a	Compare ACC with the content of memory Example: *comp* MEM, a; Result: Flag will be changed by regarding as (MEM - a) Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

Bit Operation Instructions (6)

set0 IO.n	Set bit n of IO port to low Example: *set0* pa.5 ; Result: set bit 5 of port A to low Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set1 IO.n	Set bit n of IO port to high Example: *set1* pb.5 ; Result: set bit 5 of port B to high Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
tog IO.n	Toggle bit state of bit n of IO port

	Example: *tog* pa.5 ; Result: toggle bit 5 of port A Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set0 M.n	Set bit n of memory to low Example: *set0* MEM.5 ; Result: set bit 5 of MEM to low Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
set1 M.n	Set bit n of memory to high Example: *set1* MEM.5 ; Result: set bit 5 of MEM to high Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
swapc IO.n	Swap the nth bit of IO port with carry bit Example: *swapc* IO.0 ; Result: C ← IO.0 , IO.0 ← C When IO.0 is a port to output pin, carry C will be sent to IO.0; When IO.0 is a port from input pin, IO.0 will be sent to carry C; Affected flags: 『N』Z 『Y』C 『N』AC 『N』OV Application Example1 (serial output) : --- … set1 pac.0 ; // set PA.0 as output … set0 flag.1 ; // C=0 swapc pa.0 ; // move C to PA.0 (bit operation), PA.0=0 set1 flag.1 ; // C=1 swapc pa.0 ; // move C to PA.0 (bit operation), PA.0=1 … --- Application Example2 (serial input) : --- … set0 pac.0 ; // set PA.0 as input … swapc pa.0 ; // read PA.0 to C (bit operation) src a ; // shift C to bit 7 of ACC swapc pa.0 ; // read PA.0 to C (bit operation) src a ; // shift new C to bit 7, old C … ---

Conditional Operation Instructions (13)

ceqsn a, I	Compare ACC with immediate data and skip next instruction if both are equal. Flag will be changed like as (a ← a - I) Example: *ceqsn* a,0x55 ; *inc* MEM ; *goto* error ; Result: If a=0x55, then “goto error”; otherwise, “inc MEM”. Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
ceqsn a, M	Compare ACC with memory and skip next instruction if both are equal.

	Flag will be changed like as (a ← a - M) Example: *ceqsn* a, MEM; Result: If a=MEM, skip next instruction Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
ceqsn M, a	Compare ACC with memory and skip next instruction if both are equal. Example: *ceqsn* MEM, a ; Result: If a=MEM, skip next instruction Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
t0sn IO.n	Check IO bit and skip next instruction if it's low Example: *t0sn* pa.5 ; Result: If bit 5 of port A is low, skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t1sn IO.n	Check IO bit and skip next instruction if it's high Example: *t1sn* pa.5 ; Result: If bit 5 of port A is high, skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t0sn M.n	Check memory bit and skip next instruction if it's low Example: *t0sn* MEM.5 ; Result: If bit 5 of MEM is low, then skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
t1sn M.n	Check memory bit and skip next instruction if it's high EX: *t1sn* MEM.5 ; Result: If bit 5 of MEM is high, then skip next instruction Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
izsn a	Increment ACC and skip next instruction if ACC is zero Example: *izsn* a ; Result: a ← a + 1,skip next instruction if a = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
dzsn a	Decrement ACC and skip next instruction if ACC is zero Example: *dzsn* a ; Result: A ← A - 1,skip next instruction if a = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
izsn M	Increment memory and skip next instruction if memory is zero Example: *izsn* MEM ; Result: MEM ← MEM + 1, skip next instruction if MEM= 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV
dzsn M	Decrement memory and skip next instruction if memory is zero Example: *dzsn* MEM ; Result: MEM ← MEM - 1, skip next instruction if MEM = 0 Affected flags: 『Y』Z 『Y』C 『Y』AC 『Y』OV

wait0 IO.n	Go next instruction until bit n of IO port is low, otherwise, wait here. Example: *wait0* pa.5; Result: Wait bit 5 of port A low to execute next instruction; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
wait1 IO.n	Go next instruction until bit n of IO port is high, otherwise, wait here. Example: *wait1* pa.5; Result: Wait bit 5 of port A high to execute next instruction; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

System control Instructions (18)

call label	Function call, address can be full range address space Example: *call* function1; Result: [sp] ← pc + 1 pc ← function1 sp ← sp + 2 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
goto label	Go to specific address which can be full range address space Example: *goto* error; Result: Go to error and execute program. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
icall [M]	Index call: Function call which addressed by the content of memory, It needs 2T to execute this instruction. Example: *icall* [MIDX]; Result: Call function and the address specified by the content of MIDX Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: -- word ptr1 ; // declare a RAM pointer ... mov a, la@icall_routine ; mov lb@ptr1, a ; // move icall_routine low address to RAM pointer (LSB) mov a, ha@icall_routine ; mov hb@ptr1, a ; // move icall_routine high address to RAM pointer (MSB) icall ptr1 ; // indirect call icall_routine ... ... icall_routine: ... ... ret ; --

igoto [M]	Index goto: Go to address that specified by the content of memory. Example: *igoto* [error]; Result: Go to address which specified the content of error Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: --- word ptr1 ; // declare a RAM pointer … mov a, la@igoto_address ; mov lb@ptr1, a ; // move igoto_address low address to RAM pointer (LSB) mov a, ha@igoto_address ; mov hb@ptr1, a ; // move igoto_address high address to RAM pointer (MSB) igoto ptr1 ; // indirect goto (igoto_address) … … igoto_address: … ---
delay I	Delay the (N+1) cycles which N is specified by the immediate data, the timing is based on the executing FPP unit. After the *delay* instruction is executed, the ACC will be zero. Example: *delay* 0x05; Result: Delay 6 cycles here Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
delay a	Delay the (N+1) cycles which N is specified by the content of ACC, the timing is based on the executing FPP unit. After the *delay* instruction is executed, the ACC will be zero. Example: *delay* a; Result: Delay 16 cycles here if ACC=0fh Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
delay M	Delay the (N+1) cycles which N is specified by the content of memory, the timing is based on the executing FPP unit. After the *delay* instruction is executed, the ACC will be zero. Example: *delay* M; Result: Delay 256 cycles here if M=ffh Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
ret I	Place immediate data to ACC, then return Example: *ret* 0x55; Result: A ← 55h ret ; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
ret	Return to program which had function call Example: *ret ;* Result: sp ← sp - 2 pc ← [sp] Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

reti	Return to program that is interrupt service routine. After this command is executed, global interrupt is enabled automatically. Example: *reti*; Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
nop	No operation Example: *nop*; Result: nothing changed Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
pcadd a	Next program counter is current program counter plus ACC. Example: *pcadd a*; Result: pc ← pc + a Affected flags: 『N』Z 『N』C 『N』AC 『N』OV Application Example: -- … mov a, 0x02 ; pcadd a ; // PC <- PC+2 goto err1 ; goto correct ; // jump here goto err2 ; goto err3 ; … correct: // jump here … --
engint	Enable global interrupt enable Example: *engint*; Result: Interrupt request can be sent to FPP0 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
disgint	Disable global interrupt enable Example: *disgint* ; Result: Interrupt request is blocked from FPP0 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
stopsys	System halt. Example: *stopsys*; Result: Stop the system clocks and halt the system Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
reset	Reset the whole chip, its operation will be same as hardware reset. Example: *reset* ; Result: Reset the whole chip. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

wdreset	Reset Watchdog timer. Example: *wdreset* ; Result: Reset Watchdog timer. Affected flags: 『N』Z 『N』C 『N』AC 『N』OV
pmode n	Operational mode selection for each FPP unit Example: *pmode* 0; Result: FPP units bandwidth sharing is set to mode 0 Mode FPP0 ~ FPP7 bandwidth sharing 0: /2, /2 1: /2, /4, /4 2: /4, /2, /4 3: /2, /4, /8, /8 4: /4, /2, /8, /8 5: /8, /2, /4, /8 6: /4, /4, /4, /4 7: /8, /4, /4, /4, /8 8: /2, /8, /8, /8, /8 9: /4, /4, /4, /8, /8 10: /8, /2, /8, /8, /8 11: /2, /8, /8, /8, /16, /16 12: /16, /2, /8, /8, /8, /16 13: /4, /4, /8, /8, /8, /8 14: /8, /4, /4, /8, /8, /8 15: /4, /4, /4, /8, /16, /16 16: /8, /4, /4, /4, /16, /16 17: /16, /4, /4, /4, /8, /16 18: /2, /8, /8, /16, /16, /16, /16 19: /8, /2, /8, /16, /16, /16, /16 20: /16, /2, /8, /8, /16, /16, /16 21: /4 , /4, /4, /16, /16, /16, /16 22: /16, /4, /4, /4, /16, /16, /16 23: /4, /8, /8, /8, /8, /8, /8 24: /8, /2, /16, /16, /16, /16, /16, /16 25: /4, /8, /4, /8, /16, /16, /16, /16 26: /8, /4, /4, /8, /16, /16, /16, /16 27: /2, /8, /16, /16, /16, /16, /16, /16 28: /4, /4, /8, /8, /16, /16, /16, /16 29: /16, /2, /8, /16, /16, /16, /16, /16 30: /8, /4, /4, /8, /16, /16, /16, /16 31: /8, /8, /8, /8, /8, /8, /8, /8 Affected flags: 『N』Z 『N』C 『N』AC 『N』OV

Summary of Instructions Execution Cycle

2T	ldtabh, ldtabl, idxm, icall, pushw, popw
1T	Others

Summary of affected flags by Instructions

Instruction	Z	C	AC	OV	Instruction	Z	C	AC	OV	Instruction	Z	C	AC	OV
mov a, I	-	-	-	-	*mov* M, a	-	-	-	-	*mov* a, M	Y	-	-	-
mov a, IO	Y	-	-	-	*mov* IO, a	-	-	-	-	*nmov* M, a	-	-	-	-
nmov a, M	Y	-	-	-	*pushw* *word*	-	-	-	-	*pushw* pcN	-	-	-	-
popw *word*	-	-	-	-	*popw* pcN	-	-	-	-	*ldtabh* index		-	-	-
ldtabh index		-	-	-	*ldt16* index	-	-	-	-	*stt16* index	-	-	-	-
idxm a, index	Y	-	-	-	*idxm* index, a	-	-	-	-	*xch* M	-	-	-	-
pushaf	-	-	-	-	*popaf*	-	-	-	-	*add* a, I	Y	Y	Y	Y
add a, M	Y	Y	Y	Y	*add* M, a	Y	Y	Y	Y	*addc* a, M	Y	Y	Y	Y
addc M, a	Y	Y	Y	Y	*addc* a	Y	Y	Y	Y	*addc* M	Y	Y	Y	Y
nadd a, M	Y	Y	Y	Y	*nadd* M, a	Y	Y	Y	Y	*sub* a, I	Y	Y	Y	Y
sub a, M	Y	Y	Y	Y	*sub* M, a	Y	Y	Y	Y	*subc* a, M	Y	Y	Y	Y
subc M, a	Y	Y	Y	Y	*subc* a	Y	Y	Y	Y	*subc* M	Y	Y	Y	Y
inc M	Y	Y	Y	Y	*dec* M	Y	Y	Y	Y	*clear* M	-	-	-	-
sr a	-	Y	-	-	*src* a	-	Y	-	-	*sr* M	-	Y	-	-
src M	-	Y	-	-	*sl* a	-	Y	-	-	*slc* a	-	Y	-	-
sl M	-	Y	-	-	*slc* M	-	Y	-	-	*swap* a	-	-	-	-
swap M	-	-	-	-	*and* a, I	Y	-	-	-	*and* a, M	Y	-	-	-
and M, a	Y	-	-	-	*or* a, I	Y	-	-	-	*or* a, M	Y	-	-	-
or M, a	Y	-	-	-	*xor* a, I	Y	-	-	-	*xor* a, M	Y	-	-	-
xor M, a	Y	-	-	-	*not* a	Y	-	-	-	*not* M	Y	-	-	-
neg a	Y	-	-	-	*neg* M	Y	-	-	-	*comp* a, I	Y	Y	Y	Y
comp a, M	Y	Y	Y	Y	*comp* M, a	Y	Y	Y	Y	*set0* IO.n	-	-	-	-
set1 IO.n	-	-	-	-	*tog* IO.n	-	-	-	-	*set0* M.n	-	-	-	-
set1 M.n	-	-	-	-	*swapc* IO.n	-	Y	-	-	*ceqsn* a, I	Y	Y	Y	Y
ceqsn a, M	Y	Y	Y	Y	*ceqsn* M, a	Y	Y	Y	Y	*t0sn* IO.n	-	-	-	-
t1sn IO.n	-	-	-	-	*t0sn* M.n	-	-	-	-	*t1sn* M.n	-	-	-	-
izsn a	Y	Y	Y	Y	*dzsn* a	Y	Y	Y	Y	*izsn* M	Y	Y	Y	Y
dzsn M	Y	Y	Y	Y	*wait0* IO.n	-	-	-	-	*wait1* IO.n	-	-	-	-
call label	-	-	-	-	*goto* label	-	-	-	-	*icall* [M]	-	-	-	-
igoto [M]	-	-	-	-	*delay* I	-	-	-	-	*delay* a	-	-	-	-
delay M	-	-	-	-	*ret* I	-	-	-	-	*ret*	-	-	-	-
reti	-	-	-	-	*nop*	-	-	-	-	*pcadd* a	-	-	-	-
engint	-	-	-	-	*disgint*	-	-	-	-	*stopsys*	-	-	-	-
reset	-	-	-	-	*wdreset*	-	-	-	-	*pmode* n	-	-	-	-

Package Information

Package Marking Information

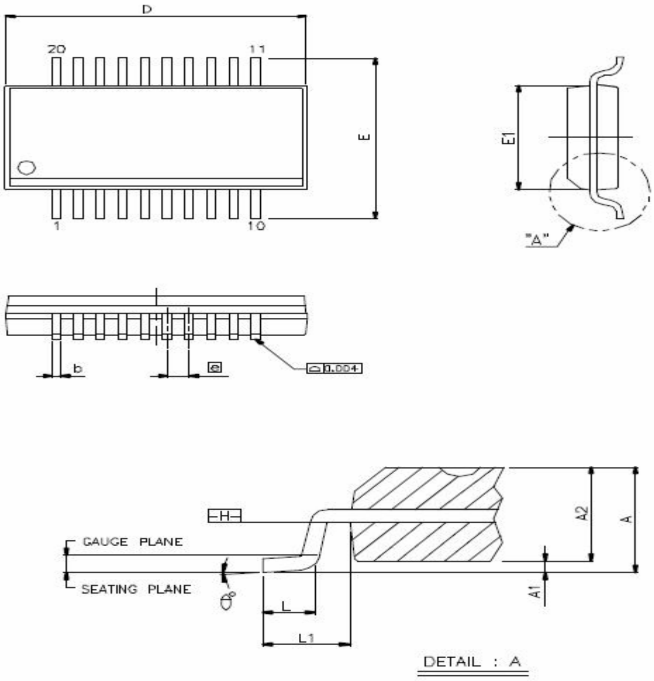
Lead of DIP

Example

Legend:

PPP…..P	**PADAUK Technology part number information**
SS……S	**Lot number information**
yy	**Year Code (last 2 digits of calendar year)**
ww	**Week Code**
XXX	**PADAUK Technology package information**

SSOP20

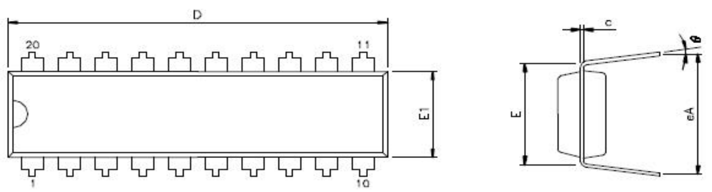


SYMBOLS	MIN.	NOM.	MAX.
A	0.053	0.064	0.069
A1	0.004	0.006	0.010
A2	–	–	0.059
b	0.008	–	0.012
C	0.007	–	0.010
D	0.337	0.341	0.344
E	0.228	0.236	0.244
E1	0.150	0.154	0.157
e	0.025 BASIC		
L	0.016	0.025	0.050
L1	0.041 BASIC		
θ°	0°	–	8°

UNIT : INCH

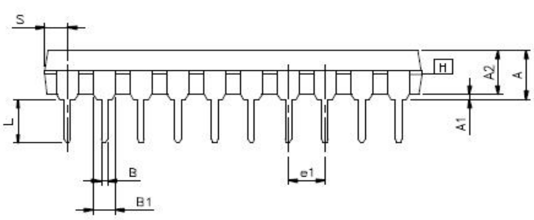
NOTES:
1 JEDEC OUTLINE : MO-137 AD
2 DIMENSION D DOES NOT INCLUDE MOLD PROTRUSIONS OR GATE BURRS MOLD PROTRUSIONS AND GATE BURRS SHALL NOT EXCEED 0.006" PER SIDE. DIMENSION E1 DOES NOT INCLUDE INTERLEAD MOLD PROTRUSIONS. INTERLEAD MOLD PROTRUSIONS SHALL NOT EXCEED 0.010" PER SIDE.
3.DIMENSION b DOES NOT INCLUDE DAMBAR PROTRUSION/INTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.004" TOTAL IN EXCESS OF b DIMENSION AT MAXIMUM MATERIAL CONDITION DAMBAR INTRUSION SHALL NOT REDUCE DIMENSION b BY MORE THAN 0.002" AT LEAST

DIP20



SYMBOLS	MIN	NOM	MAX
A	–	–	0.175
A1	0.015	–	–
A2	0.125	0.130	0.135
B	0.016	0.018	0.020
B1	0.058	0.060	0.064
c	0.008	0.010	0.011
D	1.012	1.026	1.040
E	0.290	0.300	0.310
E1	0.245	0.250	0.255
e1	0.090	0.100	0.110
L	0.120	0.130	0.140
θ	0	–	15
eA	0.335	0.355	0.375
S	–	–	0.075

UNIT : INCH



NOTES:
1.JEDEC OUTLINE : MS-001 AD
2 "D","E1" DIMENSIONS DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS.MOLD FLASH OR PROTRUSIONS SHALL NOT EXCEED .010 INCH.
3.eA IS MEASURED AT THE LEAD TIPS WITH THE LEADS UNCONSTRAINED.
4.POINTED OR ROUNDED LEAD TIPS ARE PREFERRED TO EASE INSERTION.
5.DISTANCE BETWEEN LEADS INCLUDING DAM BAR PROTRUSIONS TO BE .005 INCH MININUM.
6.DATUM PLANE H COINCIDENT WITH THE BOTTOM OF LEAD, WHERE LEAD EXITS BODY.