ICmic
IC MICROSYSTEMS

APPLICATION NOTES
AN003

TEMPERATURE SENSING APPLICATION USING ICmic DAC (ICM7363/7343/7323)

Shahab A. Najmi

ICM7363, ICM7343 and ICM7323 are 12-, 10- and 8- bit low power Digital to Analog Converters (DAC). This application note describes the usage of these DAC to acquire low frequency analog data. This provides an alternative method to acquiring data without using an expensive Analog to Digital Converter (ADC).

This option is worth considering in many applications that are typically implemented with an ADC since it can result in lower power consumption, reduced system cost and more flexibility. This approach is particularly useful if one or more of the following are true.

- Repetitive high bandwidth transient signal is to be measured. The advantage in such a case is that we can replace an expensive, high bandwidth ADC with a much cheaper DAC and comparator combination. Only the comparator has to be able to sustain the entire bandwidth of the signal while the DAC can have a long settling time without affecting the quality of the measurement. However if the signal is single shot event then the ADC method is preferable.

- DAC is already being used in the system for some other application and it can be utilized to double as an ADC when it is not being used as a DAC, thus removing the need for a separate ADC. This would help reduce the system cost and form factor.

- Low frequency data is to be measured with high accuracy and low system cost. Using a DAC might be desirable for the reason that it can sometimes reduce the amount of overhead on the processor if the processing part can be moved in the analog domain. An example of this case is when low battery alarm has to be generated, it might be desirable to use a simple DAC to provide reference (negative input) and a comparator comparing the battery voltage (positive input). The processor needs to react only in the event that the comparator output goes low thus triggering the low battery alarm.

Apart from the above mentioned cases and examples there are many cases where using a DAC is a better option than the conventional method of using an ADC. This application note gives an example of persuading the DAC to act as an accurate ADC for temperature sensing.

Temperature Sensing using a DAC

This is an application that is typically implemented with an analog to digital converter (ADC). However this note provides an alternative method of implementing such a design. This approach is particularly suitable in many cases, some of which have been mentioned in the previous section.

We can implement this system as in the block diagram below.

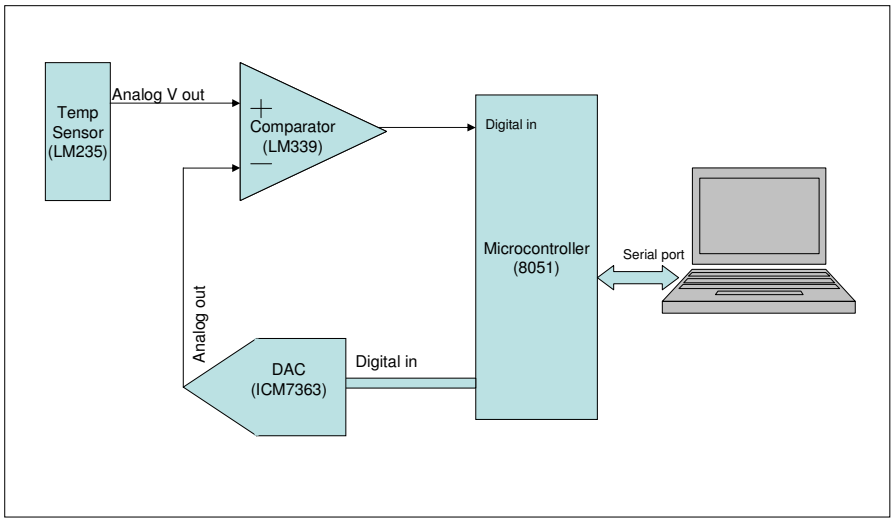

Figure 1: Digitizing temperature sensor output using ICM7363 DAC

The DAC / comparator combination can act as a successive approximation ADC. This can be done in a number of ways but the quickest is by using binary trial of weights as described below.

Successive Approximation Method

In order to digitize an analog value each bit starting from the most significant one is first set to 1 and the corresponding analog value is compared with the actual value. If the actual value is greater than the resulting value then the bit is set to one otherwise it is reset to 0. The process is repeated for all the bits till we reach the least significant bit. The algorithm for this conversion is given below.

Table 1: Algorithm for Digital conversion of data

```
Begin:

Make the most significant bit high;

Loop:
        Convert using the current value;
        Wait to allow the DAC to settle;

       If comparator input is high
                Leave the bit high;
        Else
                Make the bit low;

        Make the next bit high;
```



```
        Loop till all bits are tested;

Value after the loop would be the converted value;
```

Thus this process makes the minimum time for one complete conversion for the resulting successive approximation “ADC” equal to the conversion time of the DAC times the number of bits in DAC.

Total Conversion time (min.) = Settling time for DAC x Number of bits in DAC

Thus for 8-bit digital conversion the maximum rate for the conversion is equal to 1/8 of the DAC conversion rate. This is fast enough for many low frequency applications like the one discussed in this application note.

A simplified illustration of the operation of this algorithm is shown in the figure below.

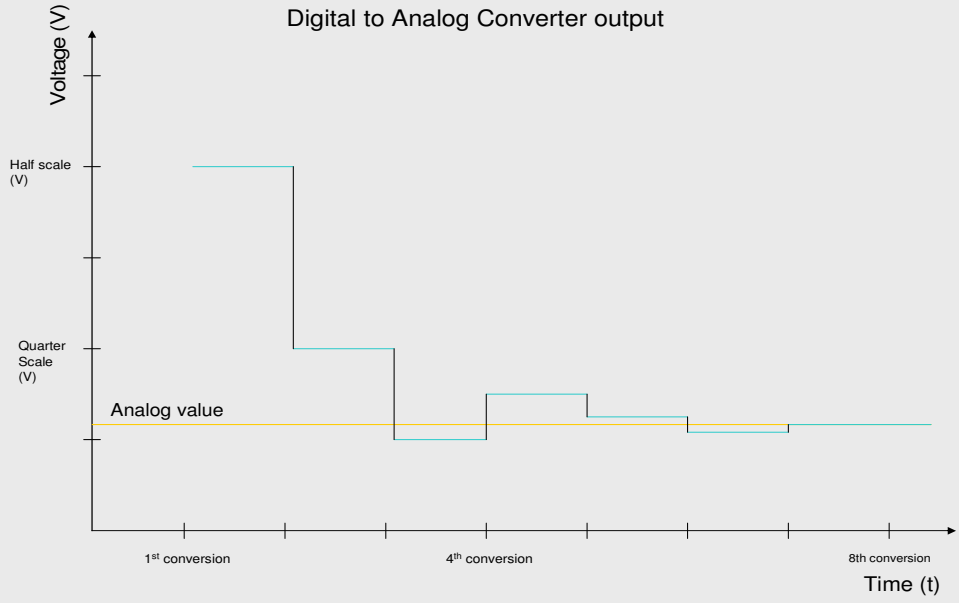

Figure 2: DAC output zeroing on analog value

Implementation for 8051 based microcontroller

The following code is for the conversion of the analog data into digital utilizing a 8051 based microcontroller.

This example uses ICM7363 which is a 12-bit low power DAC to get a 12 bit digital value for the input voltage provided by the temperature sensor.

Table 2: Sample code for successive approximation method for ICM7363

```
; Description

;           Sample code for digital conversion of analog data using ICM7363 with 8051 based microcontroller
;           This example uses port 2 to communicate with the DAC. The code provides output in ASCII character 1, 0
;           which has to be interpreted as a voltage according to the formula 2 x (Vref,in x D)/2^n

;           MCS51 pins
;                   P2.0        SCK
;                   P2.2        SDI
;                   P2.3        CSB


;           Author: Shahab A. Najmi
;           Sairan Sakrani
;           Feb, 2004


$MOD51              ; Contains Assembler specific directives

;===============================================================================
; NOTES:
; *> AT89S8252 Memory
;     (Data) RAM:    0000H – 0100H  256 bytes
;     (Data) EEPROM: 0000H – 07FFH  2K bytes (need to set EEMEN=1 with MOVX)
;     (Pgm)  Flash:  0000H – 2000H  8K bytes
;
; *> AT89S8252 implements 256 byte RAM. RAM addresses overlapping SFR are
;    accessed using indirect addressing.
;
; *> MCS-51 RAM bit addresses
;     RAM byte 0020H:  00H(LSB) – 07H(MSB)
;     RAM byte 0021H:  08H(LSB) – 0FH(MSB)
;     …
;     RAM byte 002FH:  78H(LSB) – 7FH(MSB)
;
; *> MCS-51 Hardware bit address
;     As defined in MOD51 file
;
; *> This firmware architecture
;     RAM 0000H – 0007H   (  7 bytes) R0 thru R7
;     RAM 0008H – 001FH   ( 24 bytes) Stack
;     RAM 0020H – 0100H   (224 bytes) Variables
;===============================================================================

;Hardware addresses


; SERIAL COMMUNICATION VARIABLES


P_MOSI   BIT P1.5
P_MISO   BIT P1.6
P_CLK    BIT        P1.7
SPIF     EQU 10000000B

P_SCK    EQU        P2.0                ;P2.0 – SCK
P_CLRB   EQU        P2.1                ;P2.1 - CLRB
P_SDI    EQU        P2.2                ;P2.2 - SDI
P_CSB    EQU        P2.3                ;P2.3 - CSB

;Variables – Data For Port 0 signals
P0_D0    BIT        000H                ;RAM 20.0
P0_D1    BIT        001H                ;RAM 20.1

;Variables – Data For Port 1 signals
ICLK     BIT        008H                ;RAM 21.0
IPLB     BIT        009H                ;RAM 21.1
```



```
IDS         BIT     00AH            ;RAM 21.2
mSDO        BIT     00BH            ;RAM 21.3
P1_D4       BIT     00CH            ;RAM 21.4
P1_D5       BIT     00DH            ;RAM 21.5
P1_D6       BIT     00EH            ;RAM 21.6
P1_D7       BIT     00FH            ;RAM 21.7

;Variables – Data For Port 2 signals
mSCK        BIT     010H            ;RAM 22.0
mCLRB       BIT     011H            ;RAM 22.1
mSDI        BIT     012H            ;RAM 22.2
mCSB        BIT     013H            ;RAM 22.3
ORLY        BIT     014H            ;RAM 22.4
P2_D5       BIT     015H            ;RAM 22.5
P2_D6       BIT     016H            ;RAM 22.6
P2_D7       BIT     017H            ;RAM 22.7

;Variables – Data For Port 3 signals
P3_D0       BIT     018H            ;RAM 23.0
P3_D1       BIT     019H            ;RAM 23.1
P3_D2       BIT     01AH            ;RAM 23.2
P3_D3       BIT     01BH            ;RAM 23.3
P3_D4       BIT     01CH            ;RAM 23.4
P3_D5       BIT     01DH            ;RAM 23.5
P3_D6       BIT     01EH            ;RAM 23.6
P3_D7       BIT     01FH            ;RAM 23.7

;Variables – Bit addressable flags
BITCOUNT        EQU     020H            ;RAM 24.0


;Variables – DAC codes
DACCL       EQU     030H            ;DAC code LOW byte
DACCH       EQU     031H            ;DAC code HIGH byte


;Temporary Variables
BLKVAR                  EQU     032H
TEMPVAL                 EQU     033H            ;Temporary value holder
TEMPCNT                 EQU     034H            ;Temporary control sequence holder
COMPRESULT              BIT     035H            ;Result of comparison
WDLYH                   EQU     036H            ;Delay – high byte
WDLYL                   EQU     037H            ;Delay – low byte
BLKVAR2                 EQU     038H
SENDBUF                 EQU     039H
TEMPBUF                 EQU     03AH
TEMPBUF2                EQU     03BH
;=========================================================================
;Interrupt Vectors
ORG     0000H
        LJMP    BEGIN           ;0000H - RESET vector
ORG     0003H
        LJMP    ISEXT0          ;0003H - External 0
ORG     000BH
        LJMP    ISTMR0          ;000BH - Timer/Counter 0
ORG     0013H
        LJMP    ISEXT1          ;0013H – External 1
ORG     001BH
        LJMP    ISTMR1          ;001BH – Timer/Counter 1
ORG     0023H
        LJMP    ISSER           ;0023H – Serial Port

ISEXT0:
ISTMR0:
ISEXT1:
ISTMR1:
ISSER:  RETI                    ;Empty interrupt services

;=========================================================================
;Beginning of Main Routine
```



```
;=========================================================================
ORG     0100H

BEGIN:
        MOV     IE,#00H             ;Disable interrupts


;----- Initialize interface to DACs
        CLR     P_CLRB              ;CLRB=LOW
        SETB    P_CSB               ;CSB(P2.3)=HIGH
        CLR     P_SCK               ;SCK(P2.0)=LOW
        CLR     P_SDI               ;SDI(P2.2)=LOW
        SETB    P_CLRB              ;CLRB(P2.1)=HIGH
        SETB P3.2                   ;TO BE USED AN INPUT
        MOV A, #0
        MOV BLKVAR, #0
        MOV BLKVAR2, #0
        MOV DACCL, #0FFH
        MOV DACCH, #08FH
        LCALL SNDDAC


        MOV     DACCL, #00H                 ;Initialize DAC lower byte
        MOV     DACCH, #0E0H

CONVERSION:

        MOV     BITCOUNT, #0BH              ;Initialize the bit counter


        CONVHS:                             ;Conversion High Byte setup
                MOV     A, #00H
                MOV     TEMPCNT, #08H
                MOV     TEMPVAL, #00H
                MOV     DACCL, #00H         ;Initialize DAC lower byte
                MOV     DACCH, #050H        ;Initialize DAC Upper byte


        CONVH:

                MOV     B,#02H              ;To divide by this to move bits
                MOV     TEMPVAL, DACCH ;Saves the old value temp
                MOV     A, DACCH
                ORL     A, TEMPCNT
                MOV     DACCH, A
                LCALL   SNDDAC
                LCALL   DL1MS
                LCALL   DL1MS
                LCALL   DL1MS
                LCALL   DL1MS
                LCALL   DL1MS

                JB      P3.2, NRESTH        ;if (pin 12==1){NRESTH}
                MOV     DACCH, TEMPVAL

        NRESTH:
                MOV     A, TEMPCNT
                DIV     AB
                MOV     TEMPCNT, A
                JZ      CONVLS
                LJMP    CONVH



        CONVLS:


                MOV     A, #80H
                MOV     TEMPCNT, A
```



```
                MOV     TEMPVAL, #00H


        CONVL:

                MOV     B,#02H          ;To divide by this to move bits
                MOV     TEMPVAL, DACCL  ;Saves the old value temp
                MOV     A, DACCL
                ORL     A, TEMPCNT
                MOV     DACCL, A

                LCALL   SNDDAC
                LCALL   DL1MS
                LCALL   DL1MS
                LCALL   DL1MS
                LCALL   DL1MS
                LCALL   DL1MS

                JB      P3.2, NRESTL    ;if (pin 12==0){NRESTL}
                MOV     DACCL, TEMPVAL

        NRESTL:
                MOV     A, TEMPCNT
                DIV     AB
                MOV     TEMPCNT, A
                JZ      ENDCONV
                LJMP    CONVL


ENDCONV:

        MOV SENDBUF, DACCH
        LCALL SERIALTRANS
        MOV SENDBUF, DACCL
        LCALL SERIALTRANS

        MOV SENDBUF, #10
        LCALL RS232
        MOV SENDBUF, #13
        LCALL RS232


        LJMP CONVERSION

;===============================================================================
;End of Main Routine
;===============================================================================


;-----   Send code to all DACs and update simultaneously
;       LCALL   SNDDAC          ;Send the high and low bytes to DAC
;
;       MOV     A,DPL
;       CJNE    A,TBENDL,SWND1
;       MOV     A,DPH           ;[12cyc]
;       CJNE    A,TBENDH,SWND2;[24cyc]

;       CLR     P2.7            ;[12cyc]
;       SETB    P2.7            ;[12cyc]
;       CLR     P3.6            ;[12cyc]
;       SETB    P3.6            ;[12cyc]
;       DJNZ    WAVEPER,SWRPT;[24cyc]
;       RET

;SWRPT:
;       MOV     DPTR,#SNTBL1+1  ;[24cyc]
;       LJMP    SWLUP           ;[24cyc]

;SWND1:
;       NOP                     ;[12cyc]
```

```
;           NOP                         ;[12cyc]
;           NOP                         ;[12cyc]
;
;SWND2:
;           NOP                         ;[12cyc]
;           NOP                         ;[12cyc]
;           NOP                         ;[12cyc]
;           NOP                         ;[12cyc]
;           NOP                         ;[12cyc]
;           NOP                         ;[12cyc]
;           NOP                         ;[12cyc]
;           NOP                         ;[12cyc]
;;          LJMP    SWLUP               ;[24cyc]


;=========================================================================
;Subroutine:        SNDDAC
;Description:       Send 16 bits of data to DAC.
;                   First, CSB is set to LOW, then 16 bits are sent,
;                   then, CSB is brought back HIGH.
;Arguments:   DACCH = Data high byte (Bit15 thru Bit8)
;                   DACCL = Data low byte (Bit7 thru Bit0)
;Trashed: Whatever trashed by subr SN8BIT
;*****************************
SNDDAC:             SETB    P_CSB           ;CSB should already HIGH initially, do it anyhow
            CLR     P_CSB                   ;CSB=LOW, initiate the data transfe
            MOV     A,DACCH
            LCALL   SN8BIT                  ;Send high byte – note the subr do the POP
            MOV     A,DACCL
            LCALL   SN8BIT                  ;Send low byte
            SETB    P_CSB                   ;CSB=HIGH, end the data transfer
            RET                             ;Done
;=========================================================================

;=========================================================================
;Subroutine:        SN8BIT
;Description:       Send 8 bits of data on P2.2 (SDI)
;                   Clock the P2.0 (SCK) as needed.
;Arguments:   A = data to be sent
;Trashed: C,R0
;*****************************
SN8BIT:  MOV    R0, #8                  ;Init loop counter
SN8LUP: CLR     P_SCK                   ;CLK=L
            RLC     A                   ;Set C=bit to be sent
            MOV     P_SDI,C             ;Set SDI=bit to be sent
            SETB    P_SCK               ;CLK=H, ICM7363 latches data on SDI at +ve edge

            DJNZ    R0, SN8LUP          ;Decrement counter, loop if more bits to sent
            CLR     P_SCK               ;CLK=L
            RET                         ;Done
;=========================================================================



;=========================================================================
;Subroutine:        DL1MS
;Description:       Delay about 1ms ( for crystal @7.3728 MHz )
;                   Each ms needs about 0.001sec/7.3728e+6 = 7372.8 = 7372 cycles
;Arguments:   None
;Trashed: R5, R6, R7
;*****************************
DL1MS:   MOV    R5,#150                 ;12 cyc; Init delay counter
            MOV     R6,#3               ;12 cyc
            MOV     R7,#1               ;12 cyc
;
DL1MSL: DJNZ    R5, DL1MSL              ;24 cyc         (0 + (3+0*150-1)*150) x 24
            DJNZ    R6, DL1MSL          ;24 cyc         (3 + 0*150) x 24
            DJNZ    R7, DL1MSL          ;24 cyc         (1) x 24
            RET
;=========================================================================
```



```
;===============================================================================
;Subroutine:          WVDLY
;Description:         Delay about N x 1ms ( for crystal @7.3728 MHz )
;                     Each ms needs about 0.001sec/7.3728e+6 = 7372.8 = 7372 cycles
;Arguments:    WDLYH, WDLYL – delay count in ms
;Trashed: R5, R6, R7
;*****************************
WVDLY:
            MOV      A,WDLYL
            JNZ      WVDLYL
            MOV      A,WDLYH
            JZ       WVDLYE

WVDLYL:LCALL     DL1MS
            DEC      WDLYL
            MOV      A,WDLYL
            JNZ      WVDLYL

            MOV      A,WDLYH
            JZ       WVDLYE
            DEC      WDLYH

            SJMP     WVDLYL

WVDLYE:              RET
;===============================================================================


;====================================================================
; Subroutine:          SERIALOUT
; Description:         WRITES SERIAL DATA OUT THROUGH SCL, MOSI, MISO
;                      A HAS INPUT DATA
; Trashes:             A, SPDR
;====================================================================
SERIALOUT:

            SETB P_MOSI
            SETB P_MISO
            SETB P_CLK
            MOV SPCR, #01010111B
            ;MOV A, SENDBUF



            MOV SPDR, SENDBUF
BBB:
            MOV A, SPSR
            ANL A, #SPIF
            JZ BBB

            RET

;WRITE TO SERIAL SBUF WITH RS232

;Subroutine RS232

RS232:
            MOV TMOD, #20H
            MOV TH1, #-2
            MOV SCON, #50H
            SETB TR1

TRANS:
            MOV SBUF, SENDBUF
            ;MOV SBUF, #'A'
HERE:
            JNB TI, HERE
            CLR TI
```



```
        RET
;===============================================================================
; Subroutine:       SERIALTRANS
; Description:      Transmits each bit of the conversion separately in ASCII to the serial buffer
; Trashes:          A, Whatever is trashed by RS232
;===============================================================================

SERIALTRANS:

        MOV TEMPBUF, SENDBUF
SERIALTRANS0:
        MOV A, TEMPBUF
        ANL A, #10000000B
        JZ CALLTRANSZERO0
        LJMP CALLTRANSONE0
CALLTRANSZERO0:
        LCALL TRANSZERO
        LJMP SERIALTRANS1
CALLTRANSONE0:
        LCALL TRANSONE
        LJMP SERIALTRANS1

SERIALTRANS1:
        MOV A, TEMPBUF
        ANL A, #01000000B
        JZ CALLTRANSZERO1
        LJMP CALLTRANSONE1
CALLTRANSZERO1:
        LCALL TRANSZERO
        LJMP SERIALTRANS2
CALLTRANSONE1:
        LCALL TRANSONE
        LJMP SERIALTRANS2

SERIALTRANS2:
        MOV A, TEMPBUF
        ANL A, #00100000B
        JZ CALLTRANSZERO2
        LJMP CALLTRANSONE2
CALLTRANSZERO2:
        LCALL TRANSZERO
        LJMP SERIALTRANS3
CALLTRANSONE2:
        LCALL TRANSONE
        LJMP SERIALTRANS3

SERIALTRANS3:
        MOV A, TEMPBUF
        ANL A, #00010000B
        JZ CALLTRANSZERO3
        LJMP CALLTRANSONE3
CALLTRANSZERO3:
        LCALL TRANSZERO
        LJMP SERIALTRANS4
CALLTRANSONE3:
        LCALL TRANSONE
        LJMP SERIALTRANS4

SERIALTRANS4:
        MOV A, TEMPBUF
        ANL A, #00001000B
        JZ CALLTRANSZERO4
        LJMP CALLTRANSONE4
CALLTRANSZERO4:
        LCALL TRANSZERO
        LJMP SERIALTRANS5
CALLTRANSONE4:
        LCALL TRANSONE
        LJMP SERIALTRANS5
```



```
SERIALTRANS5:
            MOV A, TEMPBUF
            ANL A, #00000100B
            JZ CALLTRANSZERO5
            LJMP CALLTRANSONE5
CALLTRANSZERO5:
            LCALL TRANSZERO
            LJMP SERIALTRANS6
CALLTRANSONE5:
            LCALL TRANSONE
            LJMP SERIALTRANS6

SERIALTRANS6:
            MOV A, TEMPBUF
            ANL A, #00000010B
            JZ CALLTRANSZERO6
            LJMP CALLTRANSONE6
CALLTRANSZERO6:
            LCALL TRANSZERO
            LJMP SERIALTRANS7
CALLTRANSONE6:
            LCALL TRANSONE
            LJMP SERIALTRANS7

SERIALTRANS7:
            MOV A, TEMPBUF
            ANL A, #00000001B
            JZ CALLTRANSZERO7
            LJMP CALLTRANSONE7
CALLTRANSZERO7:
            LCALL TRANSZERO
            LJMP SERIALTRANS8
CALLTRANSONE7:
            LCALL TRANSONE
            LJMP SERIALTRANS8

SERIALTRANS8:

            MOV SENDBUF, TEMPBUF
            RET

;=====================================================================
; Subroutines to transmit an ASCII 0 and 1
; Descrition: If a zero is to be transmitted then this subroutine sends out an ASCII zero
;                 and a ASCII 1 in case of a one
; Trashes: SENDBUF
;=====================================================================
TRANSZERO:
            MOV SENDBUF, #'0'
            LCALL RS232
            RET
TRANSONE:
            MOV SENDBUF, #'1'
            LCALL RS232
            RET
END
```


Contact Information

For any other questions / queries please feel free to contact us at support@icmic.com or contact us at the following address.

IC MICROSYSTEMS Sdn. Bhd.
Suite S06, 2320 Century Square,
63000 Cyberjaya, Selangor D.E.,
Malaysia
Tel: (603) 8319 1919
Fax: (603) 8319 1918

Disclaimer

IC MICRSOYSTEMS (ICmic) reserve the right to make corrections, modifications or changes to any of its products and services at any time and to discontinue any of its product or service without notice.

The information in this document is believed to be accurate and correct. However ICmic does not assume any liability for any loss or damage caused to any party due to the usage of the information in this document. Contents of this document are not to be released to any third party without written consent of the owner.

