MIT-WPU

School Of Electrical and Computer Engineering

T.Y. B. Tech (Trimester V)

Microcontrollers

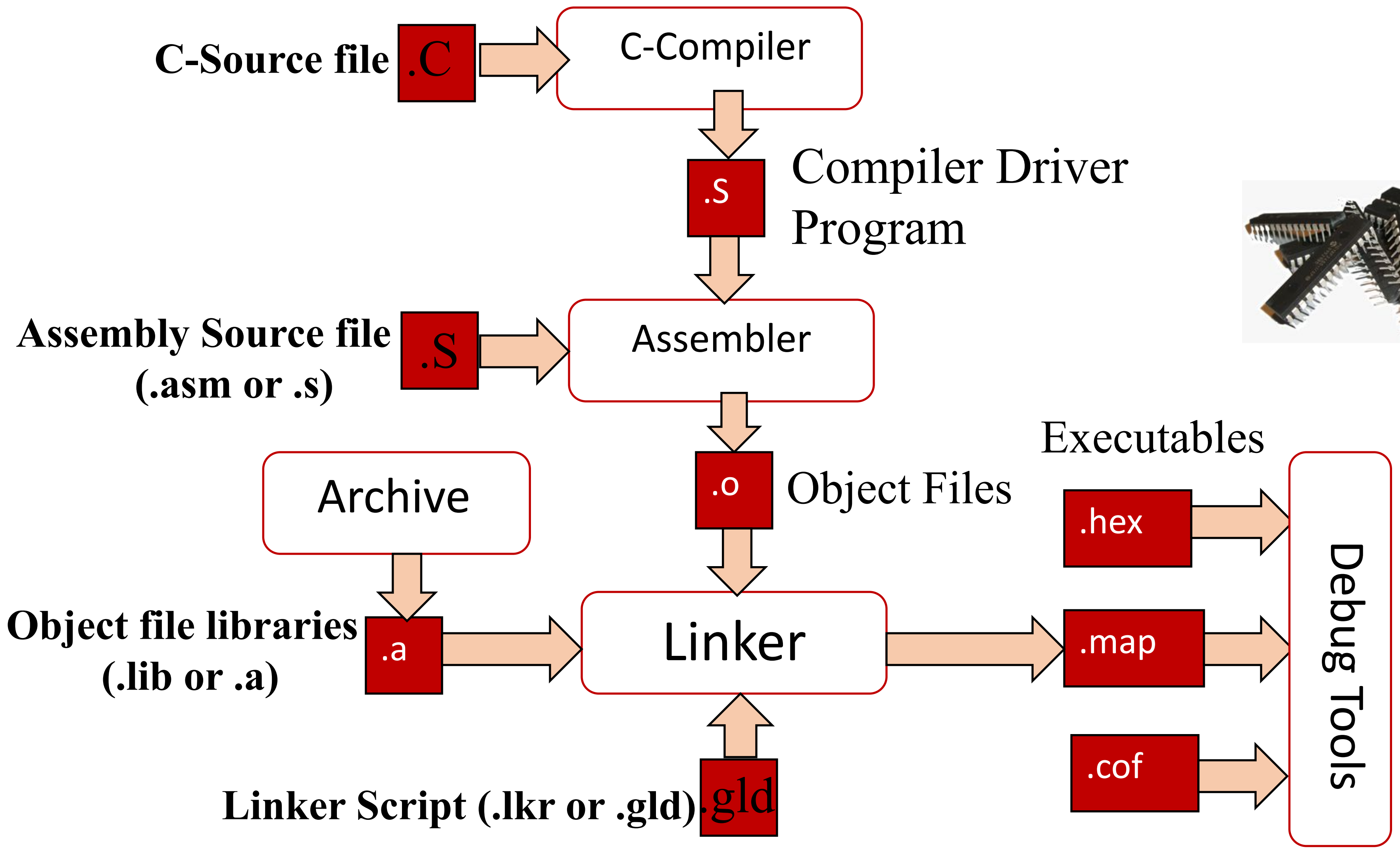
ECE2003B

Unit-III Course Content

- Interfacing of:
  - LED, Relay, Buzzer, Switch
  - 7-segment display
  - LCD
  - Keypad
  - Stepper Motor
  - DAC
  - ADC Programming
  - Programmable Counter Array (PCA)
  - DC motor control using PWM

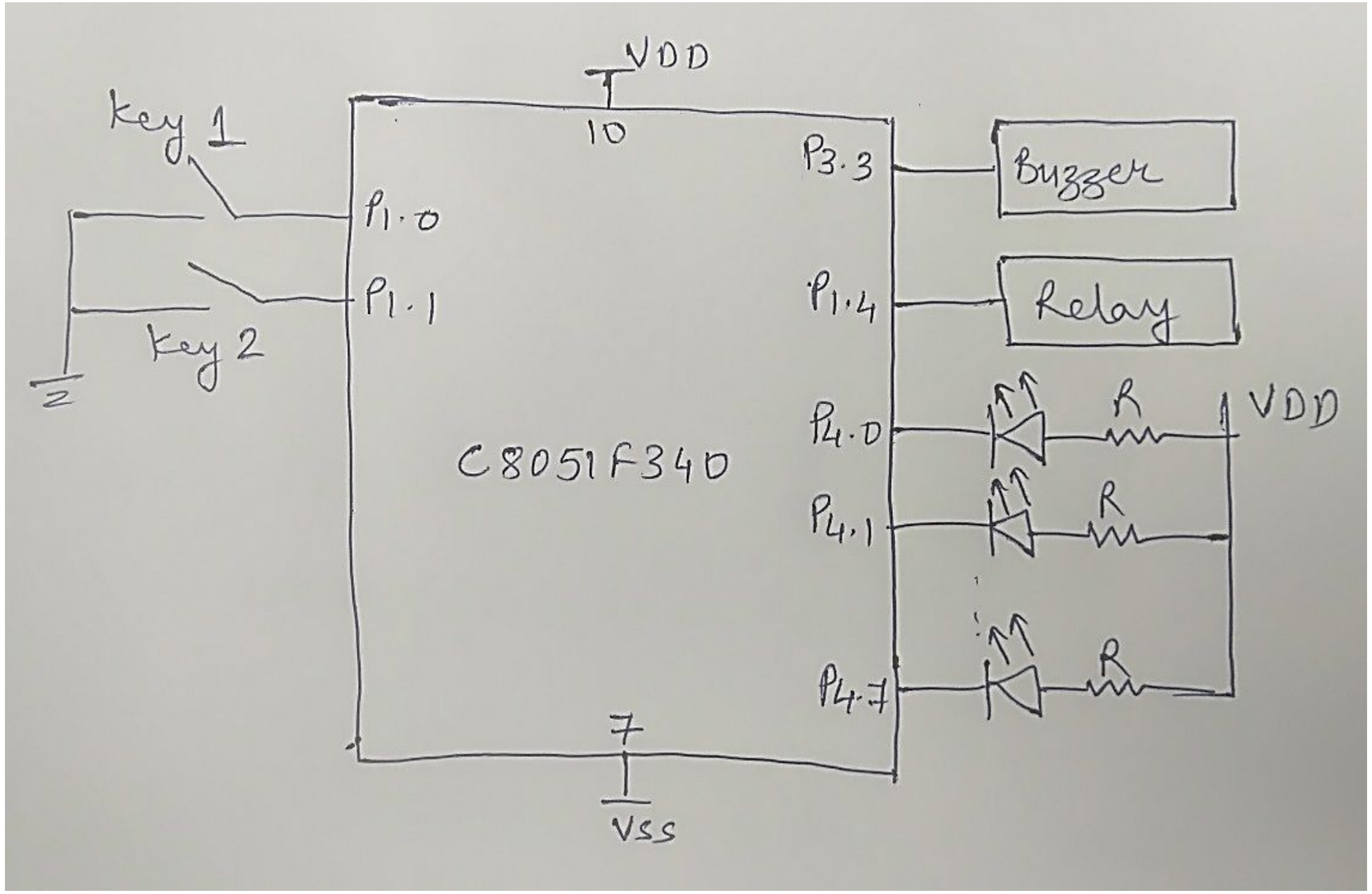
(All programs in Embedded C)

Development Tools Data Flow



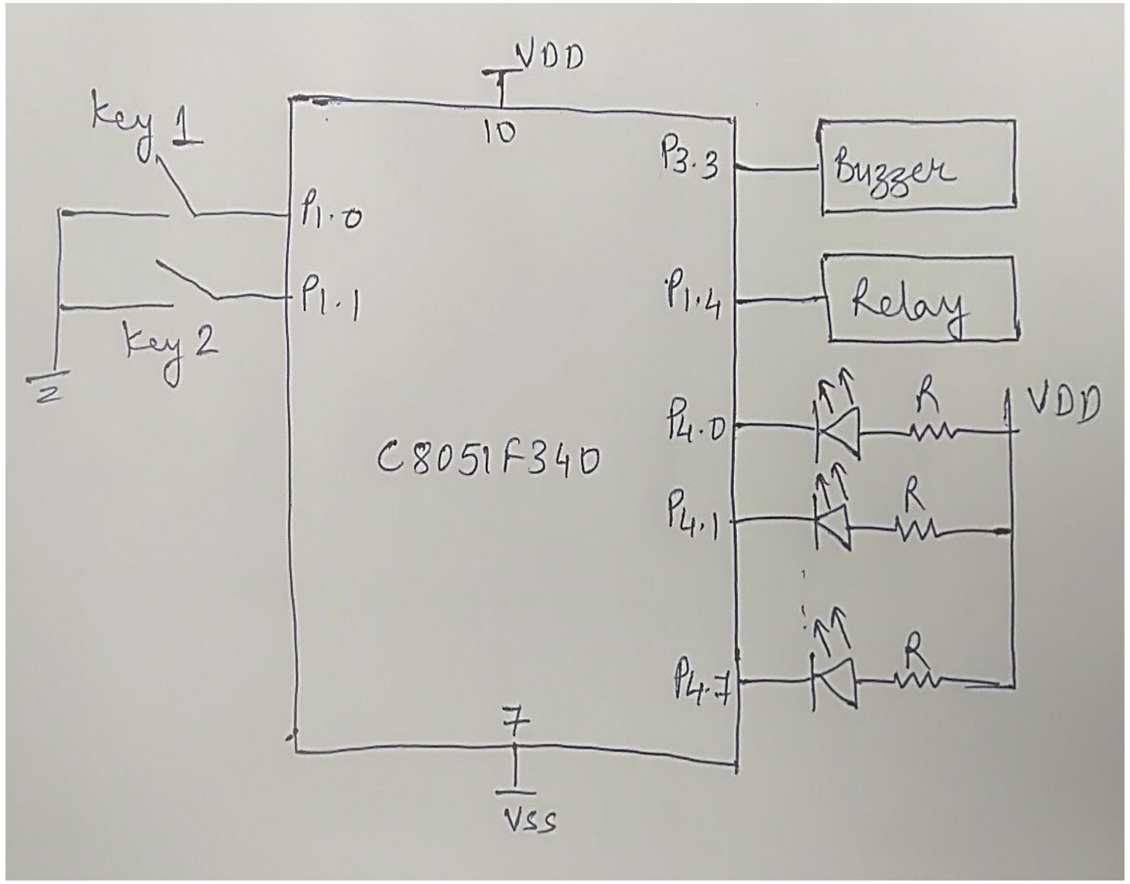
Interfacing of LED, Relay, Buzzer, Switch –

Interfacing Diagram



Interfacing of LED, Buzzer, Relay and Switch with C8051F340

- LED: LED is an output device. 8 LED's are connected to 8 port pins. The LED's are connected in common anode configuration.
  - **To turn ON the LED logic '0' must be given**
  - **To turn OFF the LED logic '1' must be given**
- Buzzer: Buzzer is an output device.
  - **To turn ON the buzzer logic '1' must be given**
  - **To turn OFF the buzzer logic '0' must be given**
- Relay: Relay is an output device.
  - **To turn ON the relay logic '0' must be given**
  - **To turn OFF the relay logic '1' must be given**



- Switch/button: Switch is an input device. Push button switches are used.
  - **When the switch is released the port pin has logic '1'**
  - **When the switch is pushed the port pin has logic '0'**

Pin Configuration

- **P1.0 &P1.1 ---- Digital Input Pins -- P1MDIN = 0xFF or 0X03**

  Key1 & Key2 ----- Logic 0 - Key Pressed

  Logic 1 – Key Released

- **P1.4 ---- Output Pin -- P1MDOUT = 0x10**

  Relay ---- ON - Logic 0

  OFF- Logic 1

- **P3.3 ---- Output Pin -- P3MDOUT = 0x08**

  Buzzer ---- ON - Logic 1

  OFF - Logic 0

- **P4 ---- Output port -- P4MDOUT = 0xFF**

  LED's ---- ON - Logic 0 ------ Common Anode

  OFF - Logic 1

Program

If KEY1 is pressed – Turn On LED, Relay and Buzzer

If KEY2 is pressed- Turn OFF LED, Relay and Buzzer

```
#include "c8051F340.h"
#define  LED  P4
sbit KEY1 = P1^0;
sbit KEY2 = P1^1;
sbit RELAY = P1^4;
sbit BUZZER = P3^3;
void main()
{
   XBR1    = 0x40;                  /* Enable Crossbar for Port 1 and 3*/
   P1MDIN = 0x03;         /* P1.0 &P1.1 pins configured as  Digital Inputs*/
   P1MDOUT = 0x10;                  /* P1.4 pins configured as Output*/

   P4MDOUT = 0xFF;                  /* All P4 pins configured as Output */

   P3MDOUT = 0x08;                  /* P3.3 pins configured as Output*/
```

Program….

If KEY1 is pressed – Turn On LED, Relay and Buzzer

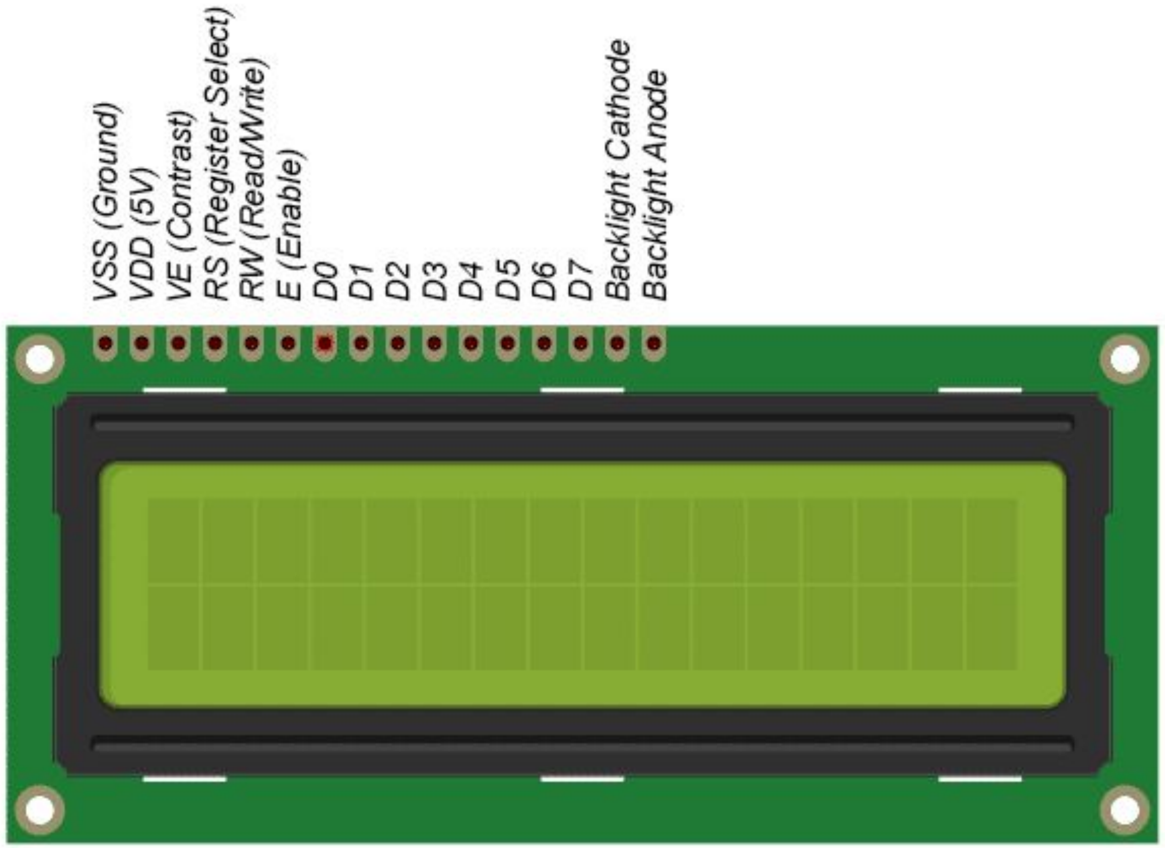
If KEY2 is pressed- Turn OFF LED, Relay and Buzzer

```
while(1)
  {
     if (KEY1 = = 0)    /* Key1 pressed*/
     {
    while(1)   /* LED, Relay, Buzzer ON */
     {
       LED = 0x00;
       RELAY = 0;
       BUZZER   = 1;
      if (KEY2 = = 0)
           {
               break;
           }
        }
     }
if (KEY2 = = 0)    /* Key1 pressed*/
    {
   while(1)   /* LED, Relay, Buzzer OFF */
    {
     LED = 0xFF;
     RELAY = 1;
     BUZZER   = 0;
    if (KEY1 = = 0)
        {
           break;
        }
      }
    }
}
```

LCD Interfacing

- Liquid Crystal Displays (LCDs)
- Cheap and easy way to display text
- Integrated controller
- The display has two register
  - Command register
  - Data register
- By Register Select(RS) you can select register for commands or data
- Data lines (D7-D0) used to transfer data and commands

Pin Description

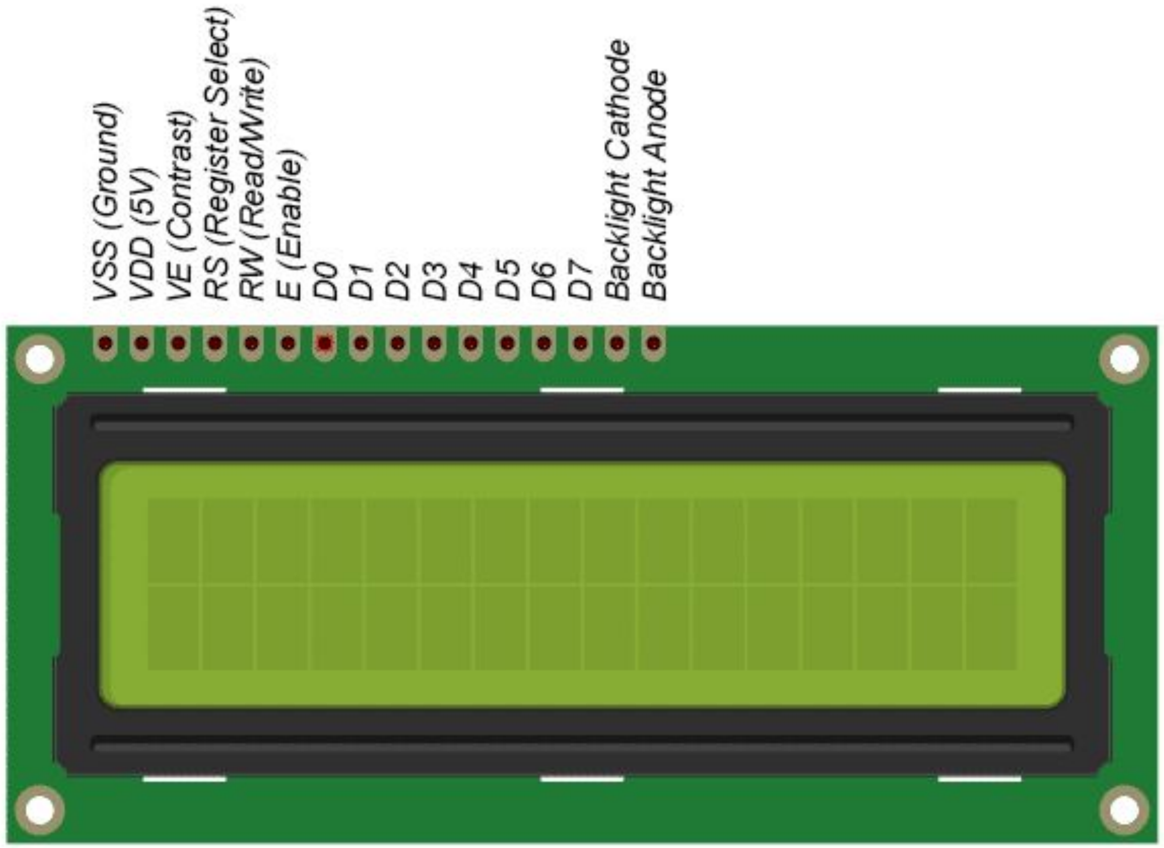


Pin No:	Name	Function
1	VSS	This pin must be connected to the ground
2	VDD	Positive supply voltage pin (5V DC)
3	VEE	Contrast adjustment
4	RS	Register selection
5	RW	Read or write
6	E	Enable
7	D0	Data
8	D1	Data
9	D2	Data
10	D3	Data
11	D4	Data
12	D5	Data
13	D6	Data
14	D7	Data
15	LED+	Back light LED+
16	LED-	Back light LED-

Alphanumeric LCD Interfacing

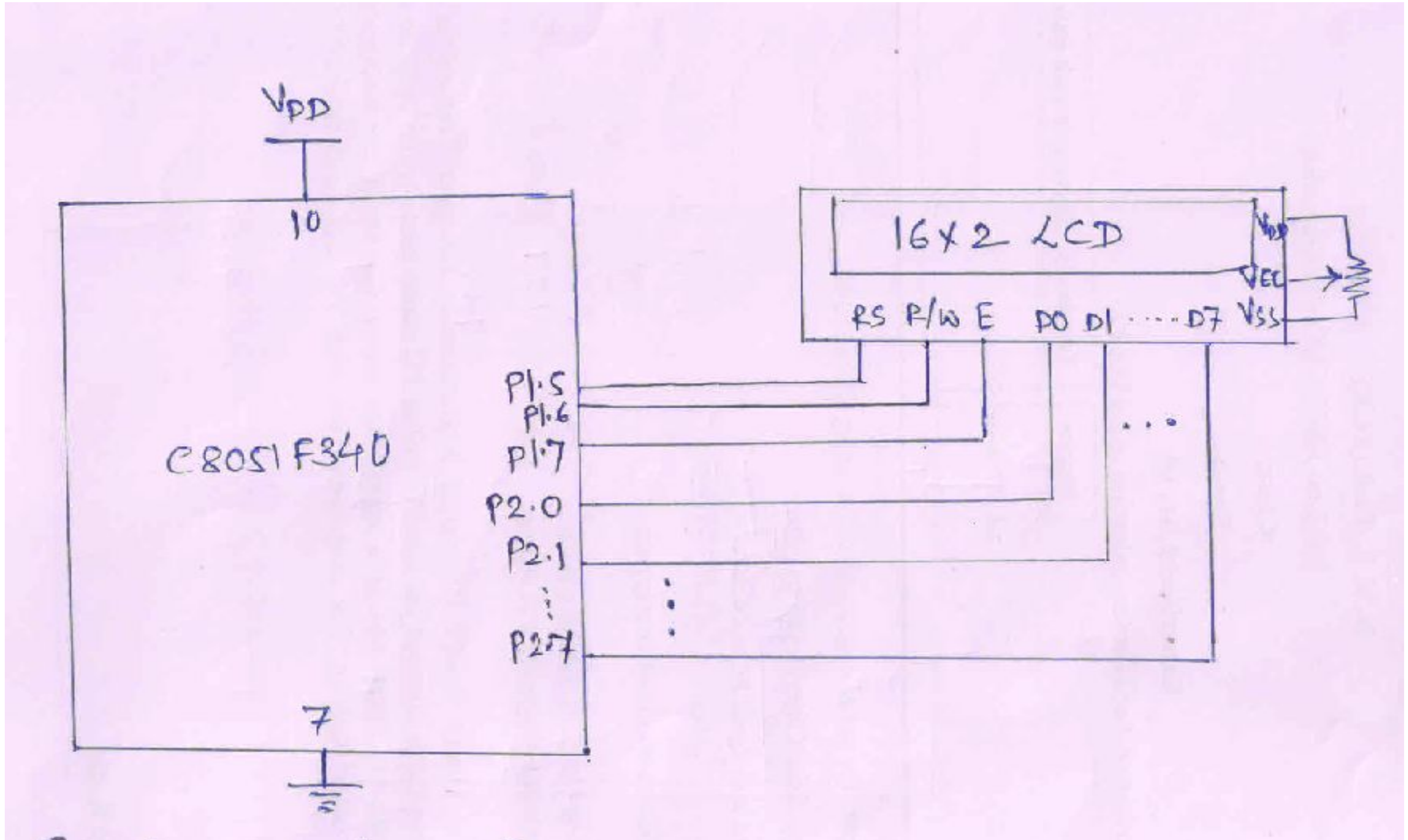
- **Pinout**
  - **8 data pins D7:D0**
  - **RS: Data or Command Register Select**
  - **RW: Read or Write**
  - **E: Enable (Latch data)**
- **RS – Register Select**
  - **RS = 0 → Command Register**
  - **RS = 1 → Data Register**
- **RW = 0 → Write , RW = 1 → Read**
- **E – Enable**
  - **Used to latch the data present on the data pins. A High to low pulse is required on Enable line**
- **D0 – D7**
  - **Bi-directional data/command pins.**
  - **Alphanumeric characters are sent in ASCII format.**
- **After writing to the LCD, it takes some time for it to complete its internal operations. During this time, it will not accept any new commands or data.**
- **We need to insert time delay between any two commands or data sent to** LCD

Command Codes



Sr.No.	Hex Code	Command to LCD instruction Register
1	01	Clear display screen
2	02	Return home
3	04	Decrement cursor (shift cursor to left)
4	06	Increment cursor (shift cursor to right)
5	05	Shift display right
6	07	Shift display left
7	08	Display off, cursor off
8	0A	Display off, cursor on
9	0C	Display on, cursor off
10	0E	Display on, cursor blinking
11	0F	Display on, cursor blinking
12	10	Shift cursor position to left
13	14	Shift cursor position to right
14	18	Shift the entire display to the left
15	1C	Shift the entire display to the right
16	80	Force cursor to beginning (1st line)
17	C0	Force cursor to beginning (2nd line)
18	38	2 lines and 5×7 matrix

Interfacing Diagram



Exp. 3 : Interfacing Digram
(LCD in 8-bit mode)

LCD Algorithm – main program

- Configure the priority crossbar decoder
- configure the ports
- Send command to select two lines, 5x7 matrix display
- Send command to clear display screen
- Send command to turn on the display and keep cursor blinking
- Send command to force the cursor to the desired position where the message is to be displayed
- Send the message character by character
- Add delay in between each command and data

LCD Algorithm – Command routine

- Select command register
- Select write operation
- Send the command to port
- Send high to low pulse on enable signal

LCD Algorithm – Data routine

- Select data register
- Select write operation
- Send the data to port
- Send high to low pulse on enable signal

LCD Algorithm – Delay routine

- Write delay function with nested “for loop”

Program for LCD (8-bit mode)

```
#include "c8051F340.h"
void DelayMs(unsigned int Ms);
void Write_Command_Lcd(unsigned char Command);
void Write_Data_Lcd(unsigned char Character);

sbit LCD_RS = P1^5;
sbit LCD_RW = P1^6;
sbit LCD_EN = P1^7;

void main()
{
XBR1    = 0x40;           /* Enable Crossbar*/
P2MDOUT = 0xFF;           /* P2 output port*/
P1MDOUT = 0xE0;           /* P1.5,P1.6&P1.7 output pins*/
   Write_Command_Lcd(0x38);     /*2 Lines and 5x7 matix*/
   DelayMs(50);                 /* Delay routine*/
   Write_Command_Lcd(0x01);     /* Clear Display Screen*/
  DelayMs(50);
  Write_Command_Lcd(0X0C);      /* Display ON, Cursor OFF*/
   DelayMs(50);
   Write_Command_Lcd(0X80);     /* Force cursor to beginning
                                   of 1st line*/
   DelayMs(50);
   Write_Data_Lcd('W');         /* Call Data routine*/
   DelayMs(50);                 /* Delay routine*/
   Write_Data_Lcd('P');
   DelayMs(50);
   Write_Data_Lcd('U');
   DelayMs(50);
 while (1);
}
```

Program for LCD (8-bit)

```
void DelayMs(unsigned int Ms)
{
        unsigned int n;
        unsigned int i;
        for (n=0; n < Ms; n++)
        {
                for (i=0; i < 65; i++);
        }
}
void Write_Command_Lcd(unsigned char Command)
{
   LCD_RS = 0;           //Select Command Register
   LCD_RW = 0;
   P2 = Command;
   LCD_EN = 1;      // EN pin high->low
   DelayMs(15);
   LCD_EN = 0;
}
```

```
void Write_Data_Lcd(unsigned char Character)
{
   LCD_RS = 1;                    //Select Date Register
   LCD_RW = 0;
   P2 = Character;
   LCD_EN = 1;             // EN pin high->low
   DelayMs(15);
   LCD_EN = 0;
}
```

Program for LCD (8-bit mode)

```
#include "c8051F340.h"
void DelayMs(unsigned int Ms);
void Write_Command_Lcd(unsigned char Command);
void Write_Data_Lcd(unsigned char Character);

sbit LCD_RS = P1^5;
sbit LCD_RW = P1^6;
sbit LCD_EN = P1^7;

void main()
{
char LCD_data[]={“MIT_WPU”};
int i;
XBR1    = 0x40;          /* Enable Crossbar*/
P2MDOUT = 0xFF;       /* P2 output port*/
P1MDOUT = 0xE0;      /* P1.5,P1.6&P1.7
                          output pins*/
Write_Command_Lcd(0x38);      /*2 Lines and 5x7 matix*/
DelayMs(50);                  /* Delay routine*/
Write_Command_Lcd(0x01);      /* Clear Display Screen*/
DelayMs(50);
Write_Command_Lcd(0x0C);      /* Display ON, Cursor OFF*/
DelayMs(50);
Write_Command_Lcd(0x80);  //Force cursor to beginning of 1st line
DelayMs(50);

for (i = 0; i != NULL; i++)       // loop to display message on LCD
   {
      Write_Data_Lcd(LCD_data[i]);
       DelayMs(50);
    }
while (1);
}
```

Program for LCD (8-bit)

```
void DelayMs(unsigned int Ms)
{
        unsigned int n;
        unsigned int i;
        for (n=0; n < Ms; n++)
        {
                for (i=0; i < 65; i++);
        }
}
void Write_Command_Lcd(unsigned char Command)
{
   LCD_RS = 0;            //Select Command Register
   LCD_RW = 0;
   P2 = Command;
   LCD_EN = 1;      // EN pin high->low
   DelayMs(15);
   LCD_EN = 0;
}
```

```
void Write_Data_Lcd(unsigned char Character)
{
   LCD_RS = 1;                      //Select Date Register
   LCD_RW = 0;
   P2 = Character;
   LCD_EN = 1;             // EN pin high->low
   DelayMs(15);
   LCD_EN = 0;
}
```

LCD 4-bit mode Interfacing Diagram

4-bit LCD only 4 data lines i.e D4-D7

- y= 0x38

- UN= y&0xf0=0x30
- Z=y &0x0f =0x08
- LN=Z<<4 =0x80

LCD (4bit Code)

```
#include "c8051F340.h"
void DelayMs(unsigned int Ms);
void Write_Command_Lcd(unsigned char Command);
void Write_Data_Lcd(unsigned char Character);

sbit LCD_RS = P1^5;
sbit LCD_RW = P1^6;
sbit LCD_EN = P1^7;

void main()
{
    XBR1    = 0x40;        /* Enable Crossbar*/
    P2MDOUT = 0xFF;
    P1MDOUT = 0xE0;
    Write_Command_Lcd(0x02);
    DelayMs(250);
    Write_Command_Lcd(0x28);
    DelayMs(250);
    Write_Command_Lcd(0x01);
    DelayMs(250);
    Write_Command_Lcd(0x0E);
    DelayMs(250);
    Write_Command_Lcd(0x80);
    DelayMs(250);
    Write_Data_Lcd('W');
    DelayMs(50);
    Write_Data_Lcd('P');
    DelayMs(50);
   Write_Data_Lcd('U');
   DelayMs(50);
    while (1);
}
```

LCD (4bit Code)

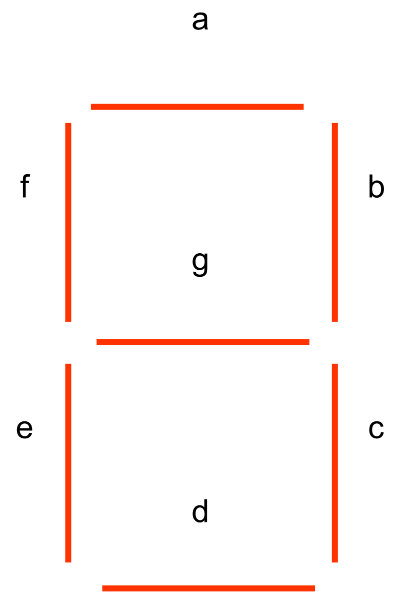
```
void DelayMs(unsigned int Ms)
{
        unsigned int n;
        unsigned int i;
        for (n=0; n < Ms; n++)
        {
          for (i=0; i < 65; i++);
        }
}
```

```
void Write_Command_Lcd(unsigned char Command)
{

   P2 =  (Command & 0xF0); // To send the upper nibble
   LCD_RS = 0;
   LCD_RW = 0;
   LCD_EN = 1;
   DelayMs(10);
   LCD_EN = 0;


   P2 = (Command & 0x0f)<<4;; // To send the lower nibble
   LCD_RS = 0;
   LCD_RW = 0;
   LCD_EN = 1;
   DelayMs(10);
   LCD_EN = 0;
   return;
}
```

```
void Write_Data_Lcd(unsigned char Character)
{
   P2 = (Character & 0xF0); // To send the upper nibble
   LCD_RS = 1;
   LCD_RW = 0;
   LCD_EN = 1;
   DelayMs(10);
   LCD_EN = 0;


   P2 = (Character & 0x0f)<<4; // To send the lower nibble
   LCD_RS = 1;
   LCD_RW = 0;
   LCD_EN = 1;
   DelayMs(10);
   LCD_EN = 0;
   return;
}
```

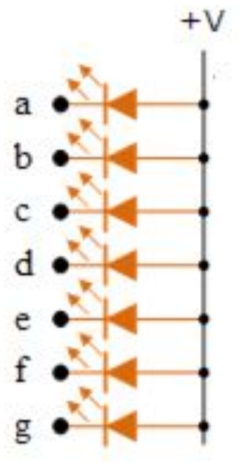
The 7-Segment Display

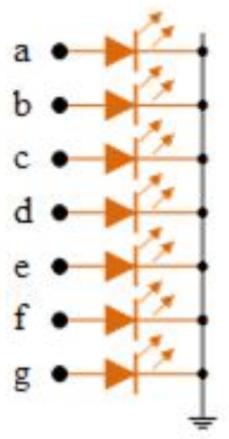
- 7 LEDs arranged to form the number 8.
- By turning on and off the appropriate segments (LEDs), different combinations can be produced.
- Useful for displaying the digits 0 through 9, and some characters.



The 7-segment Display (Cont.)

- 7-segment displays come in 2 configurations:

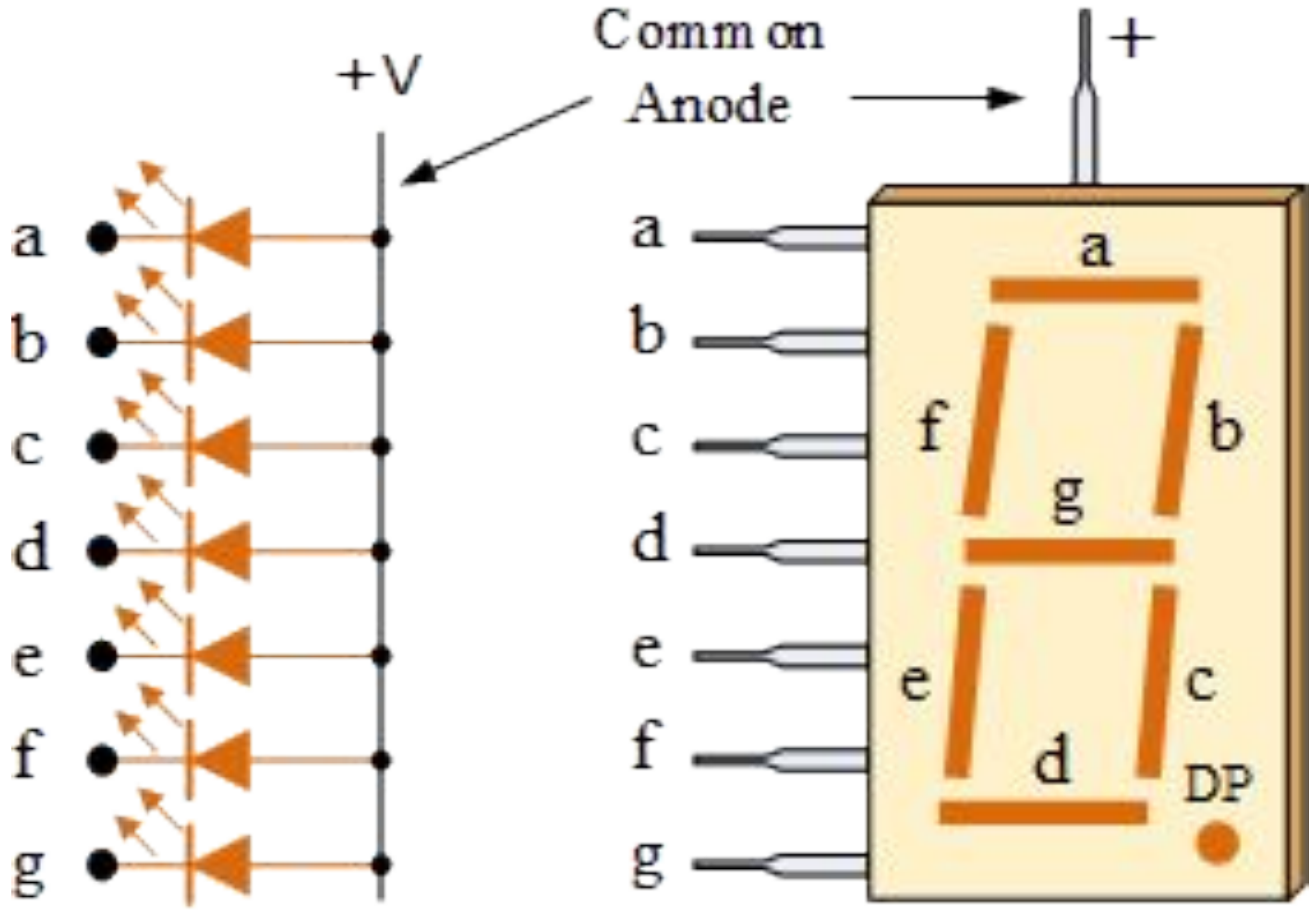




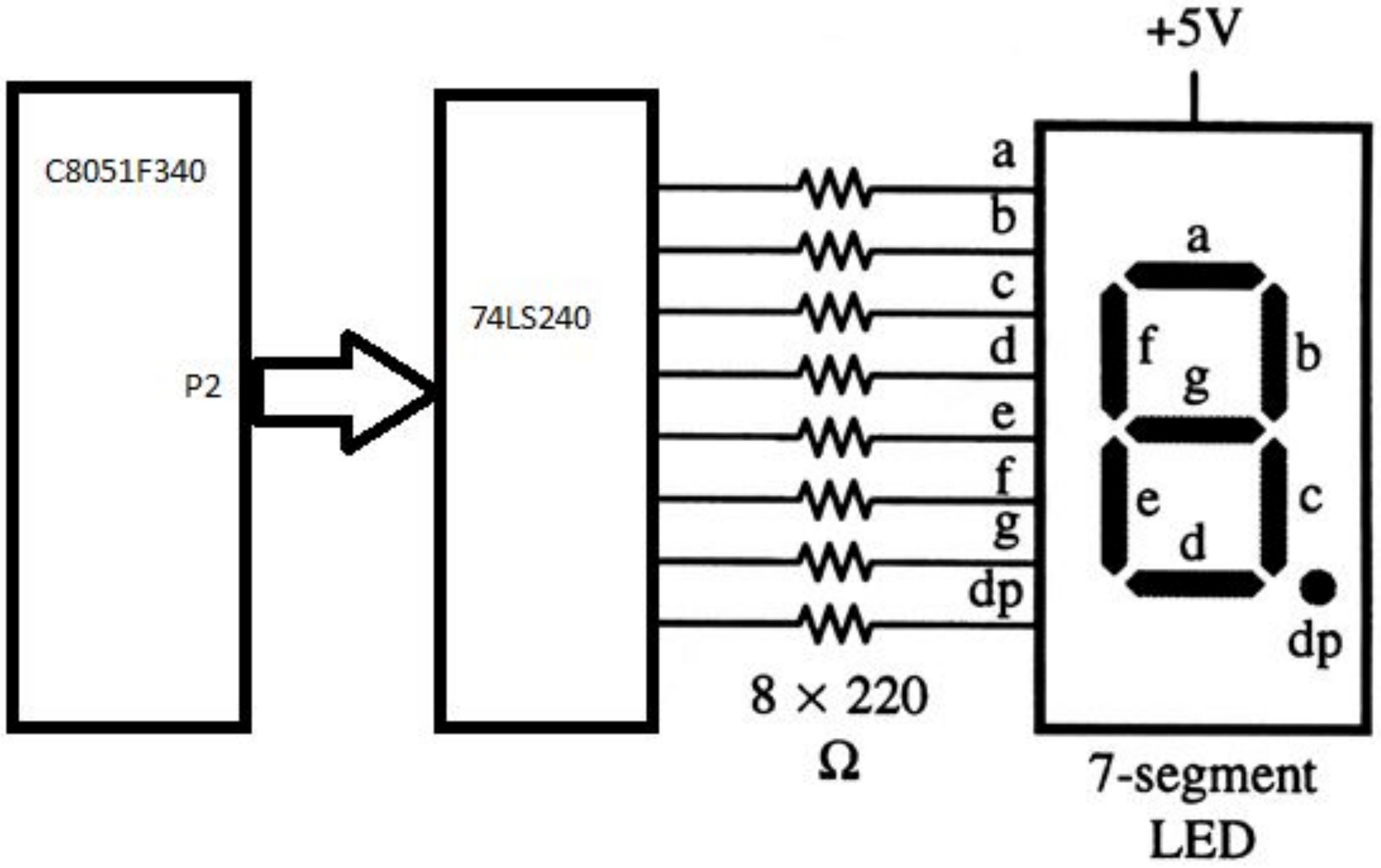
Common Anode **Common Cathode**

- It would be preferable to connect the cathode of each diode to the output pin.
- Therefore, the common anode variety would be better for our interfacing needs.

The 7-segment Display (Cont.)

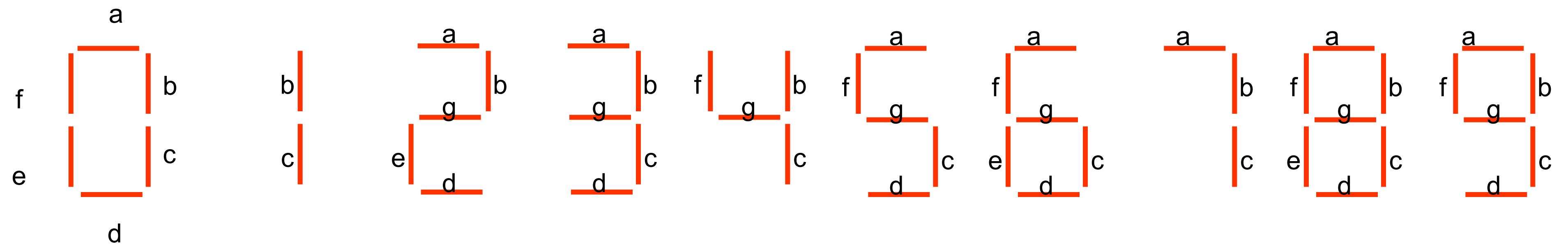


Interfacing diagram



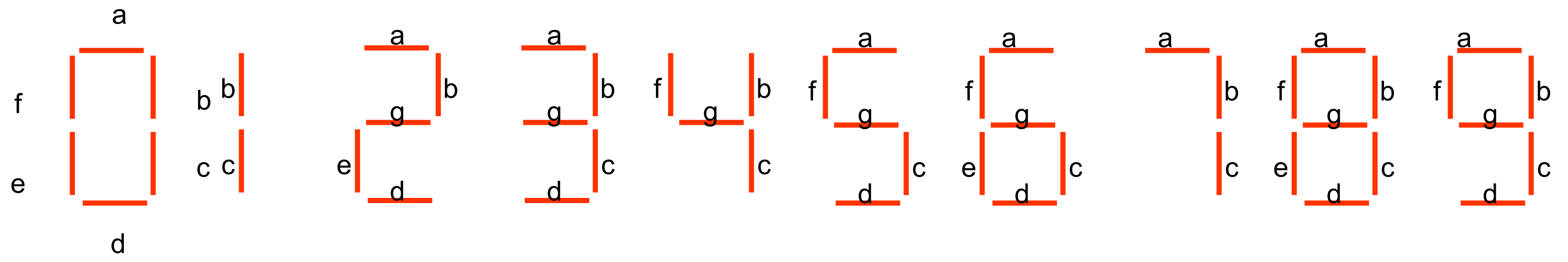
BCD to 7_Seg lookup table

Digit	h g f e d c b a 7_seg	hex
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		



BCD to 7_Seg lookup table

Digit	h g f e d c b a 7_seg	hex
0	1 1 0 0 0 0 0 0	0xC0
1	1 1 1 1 1 0 0 1	0xF9
2	1 0 1 0 0 1 0 0	0xA4
3	1 0 1 1 0 0 0 0	0xB0
4	1 0 0 1 1 0 0 1	0x99
5	1 0 0 1 0 0 1 0	0x92
6	0 1 0 0 0 0 1 0	0x82
7	1 1 1 1 1 0 0 0	0xF8
8	1 0 0 0 0 0 0 0	0x80
9	1 0 0 1 0 0 0 0	0x90



Program

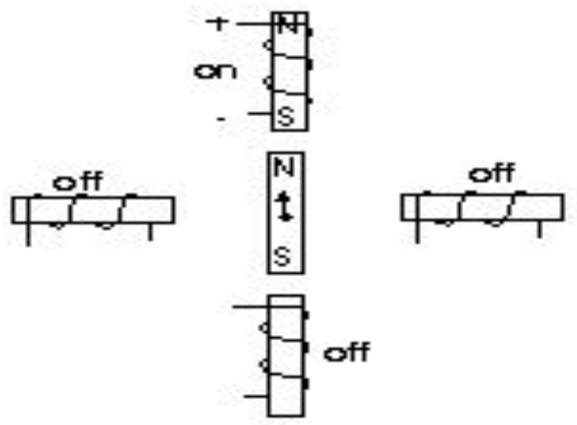
```
#include "c8051F340.h”

void DELAY_ms(unsigned int ms_Count);

main( )
{
    XBR1 = 0x40;
    P2MDOUT = 0xFF;            // All data lines Output connected to P2
    char seg_code[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
    int i;
    while (1)
    {
          for (i = 0; i <= 9; i++)        // loop to display 0-9
            {
                P2 = seg_code[i];
                DELAY_ms(1000);
            }
    }
}
```

```
void DELAY_ms(unsigned int ms_Count)
{
   unsigned int i,j;
   for(i=0;i<ms_Count;i++)
   {
      for(j=0;j<100;j++);
   }
}
```

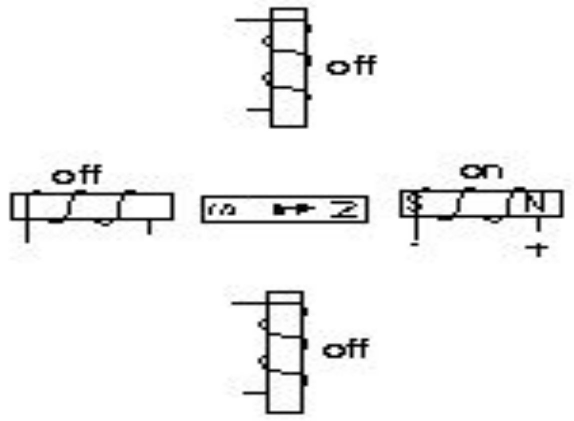
Stepper Motor

- More accurately controlled than a normal motor allowing fractional turns or ‘n’ revolutions to be easily done
- Low speed, and lower torque than a comparable D.C. motor
- Useful for precise positioning for robotics
- Servomotors require a position feedback signal for control

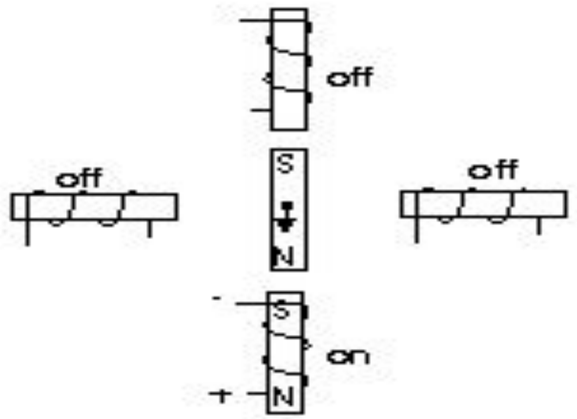
How Stepper Motor Works



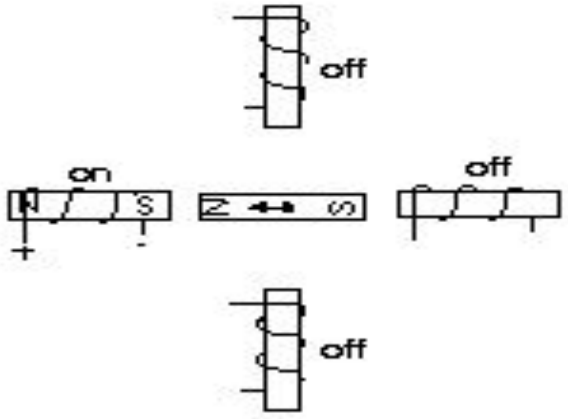
1 & 5



2



3

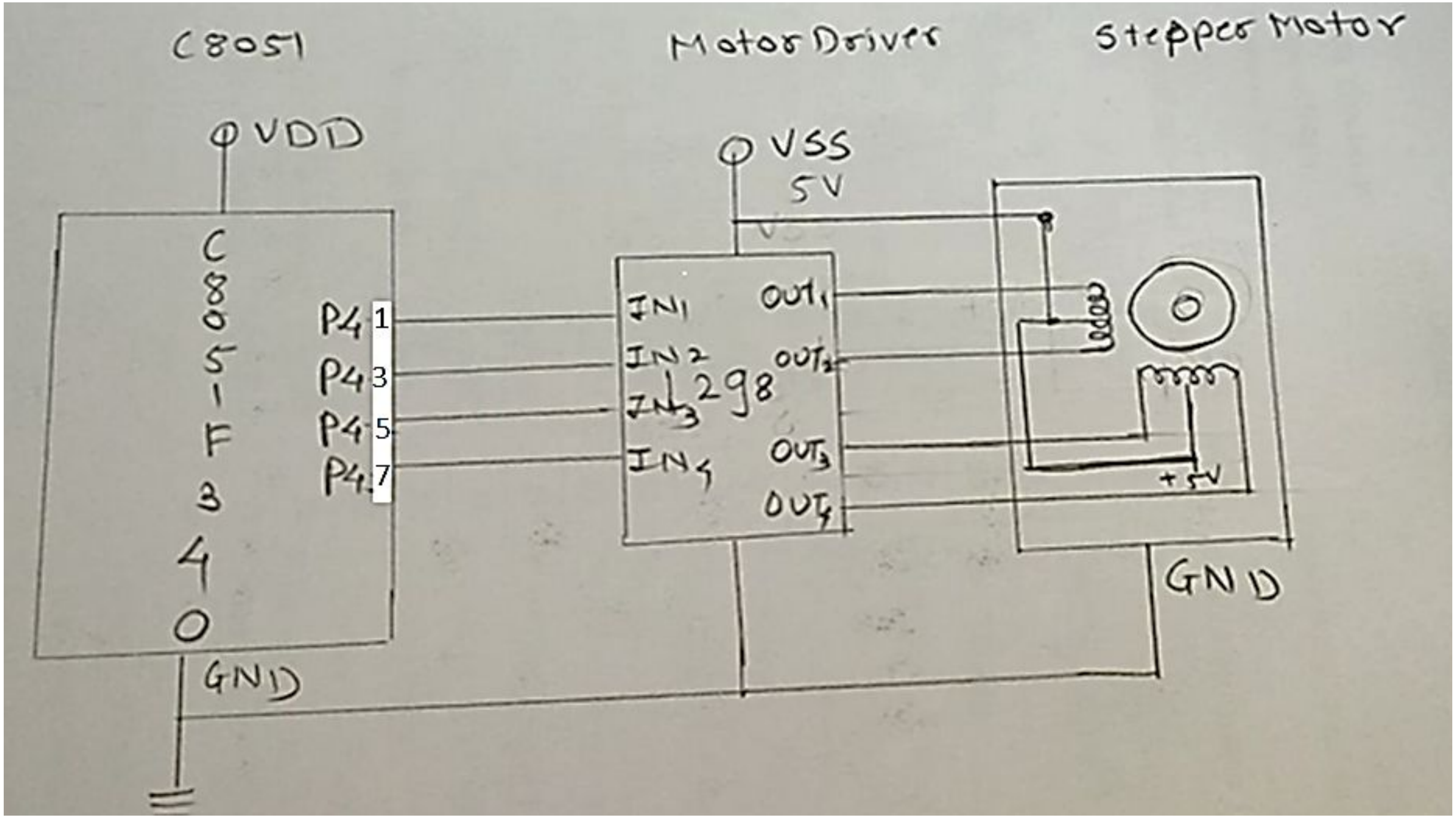


4

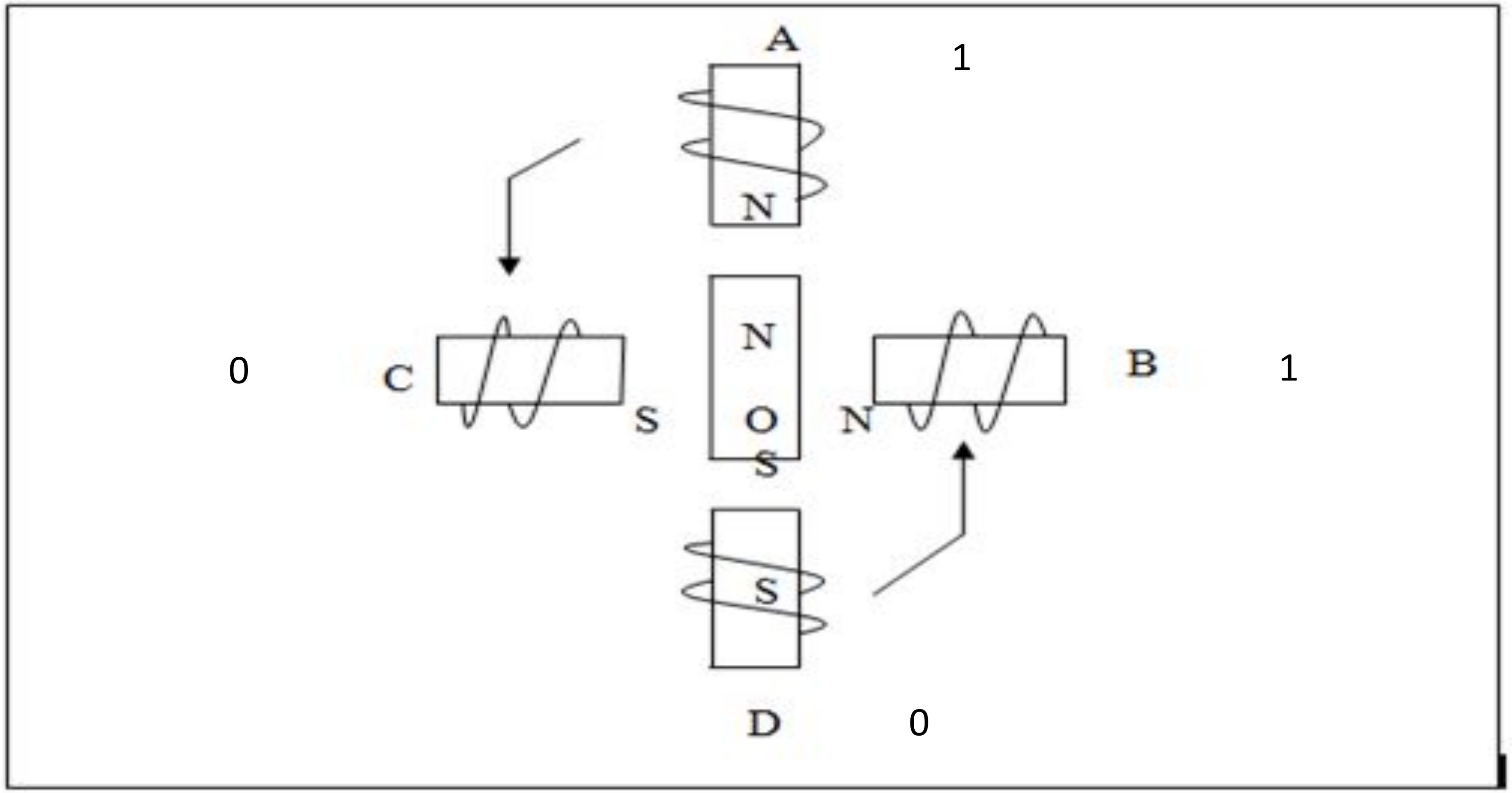
Terminology

- Steps per second, RPM
  - SPS = (RPM * SPR) /60
- Number of teeth
- Wave drive 4-step, 8-step
- Motor speed
- Holding torque

Interfacing diagram



Stepper Motor Diagram



Full Stepping

Windings	D		C		B		A	
Sequence in hex on Port 4	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
0A	0	0	0	0	1	0	1	0
88	1	0	0	0	1	0	0	0
A0	1	0	1	0	0	0	0	0
22	0	0	1	0	0	0	1	0

Program

```
#include "c8051F340.h"
void DelayMs(unsigned int Ms);
void main()
{
   P4MDOUT = 0xFF;     /* All P4 pins configured as Output*/
   while(1)
   {        P4 = 0x0A;
      DelayMs(10);
      P4 = 0x88;
      DelayMs(10);
      P4 = 0xA0;
      DelayMs(10);
      P4 = 0x22;
      DelayMs(10);   }}

void DelayMs(unsigned int Ms)
{
      unsigned int n;
      unsigned int i;
      for (n=0; n < Ms; n++)
      {
            for (i=0; i < 65; i++);
      }  }
```

Half Stepping

Windings	D		C		B		A	
Sequence in hex on Port 4	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0

Half Stepping

Windings	D		C		B		A	
Sequence in hex on Port 4	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
02	0	0	0	0	0	0	1	0
0A	0	0	0	0	1	0	1	0
08	0	0	0	0	1	0	0	0
88	1	0	0	0	1	0	0	0
80	1	0	0	0	0	0	0	0
A0	1	0	1	0	0	0	0	0
20	0	0	1	0	0	0	0	0
22	0	0	1	0	0	0	1	0

Program

```
#include "c8051F340.h“

void DelayMs(unsigned int Ms);

void main()
{
   P4MDOUT = 0xFF;        /* All P4 pins configured as Output*/

   char Half_step []={0x02,0x0A,0x08,0x88,0x80,0xA0,0x20,0x22};
   int i;
   while (1)
     {
              for (i = 0; i < 8; i++)
              {    P4 = Half_step[i];
                   DELAY_ms(10);
              }
          }
}
```

```
void DelayMs(unsigned int Ms)
{     unsigned int n;
      unsigned int i;
      for (n=0; n < Ms; n++)
      {
           for (i=0; i < 65; i++);
      }
}
```

Digital to Analog Conversion (DAC)

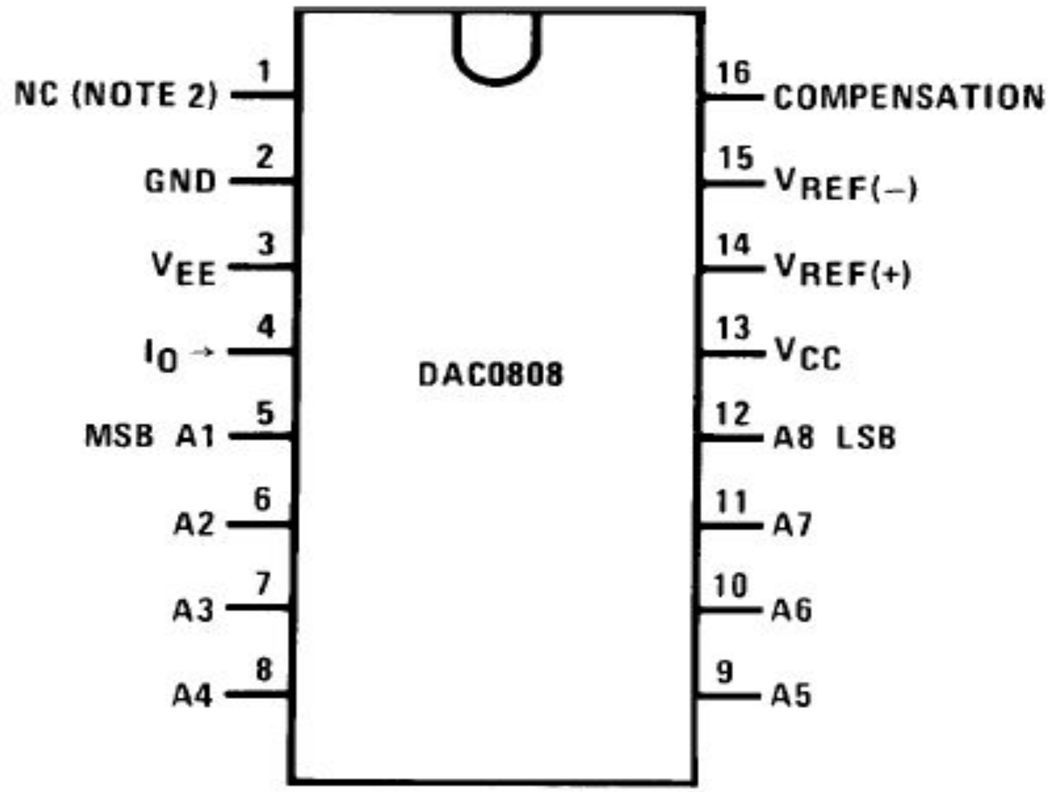
- Converts digital pulses to analog signals
- Resolution of a DAC is a function of number of binary inputs
- In DAC0808, binary numbers at D0-D7 inputs are converted to reference current I_{ref}

$$I_{out} = I_{ref}\left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256}\right)$$

D0 = MSB

D7 = LSB

DAC 0808 Pin Diagram



$$I_{out} = I_{ref}\left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256}\right)$$

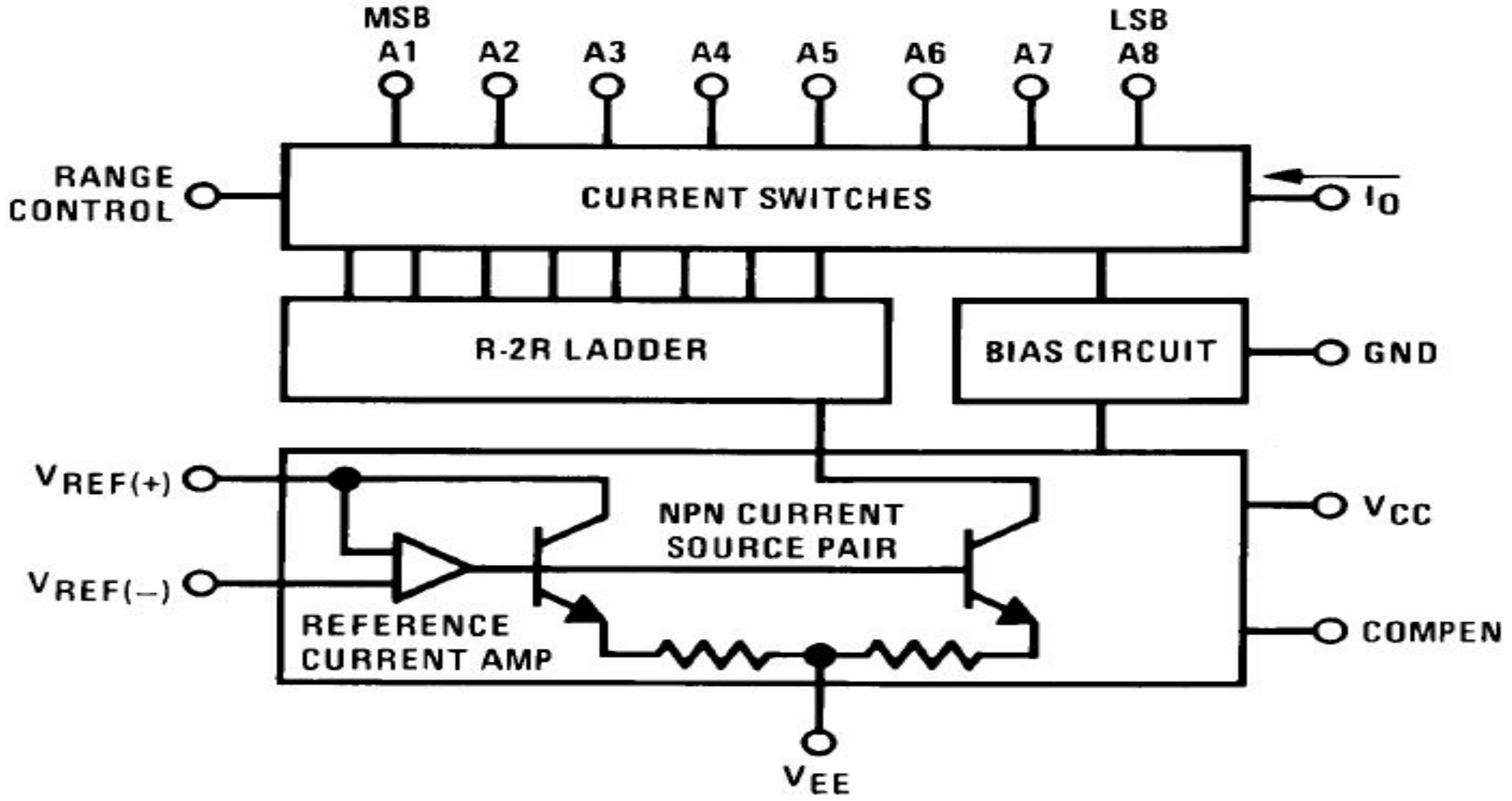
Converting I_{out} to voltage in DAC0808

- Ideally we connect the output pin I_{out} to a resistor, convert this current to voltage
- This can cause inaccuracy since the input resistance of the load where it is connected will also affect the output voltage.
- I_{ref} current output is isolated by connecting it to an op-amp such as the 741 with R_f = 5K ohms for the feedback resistor.

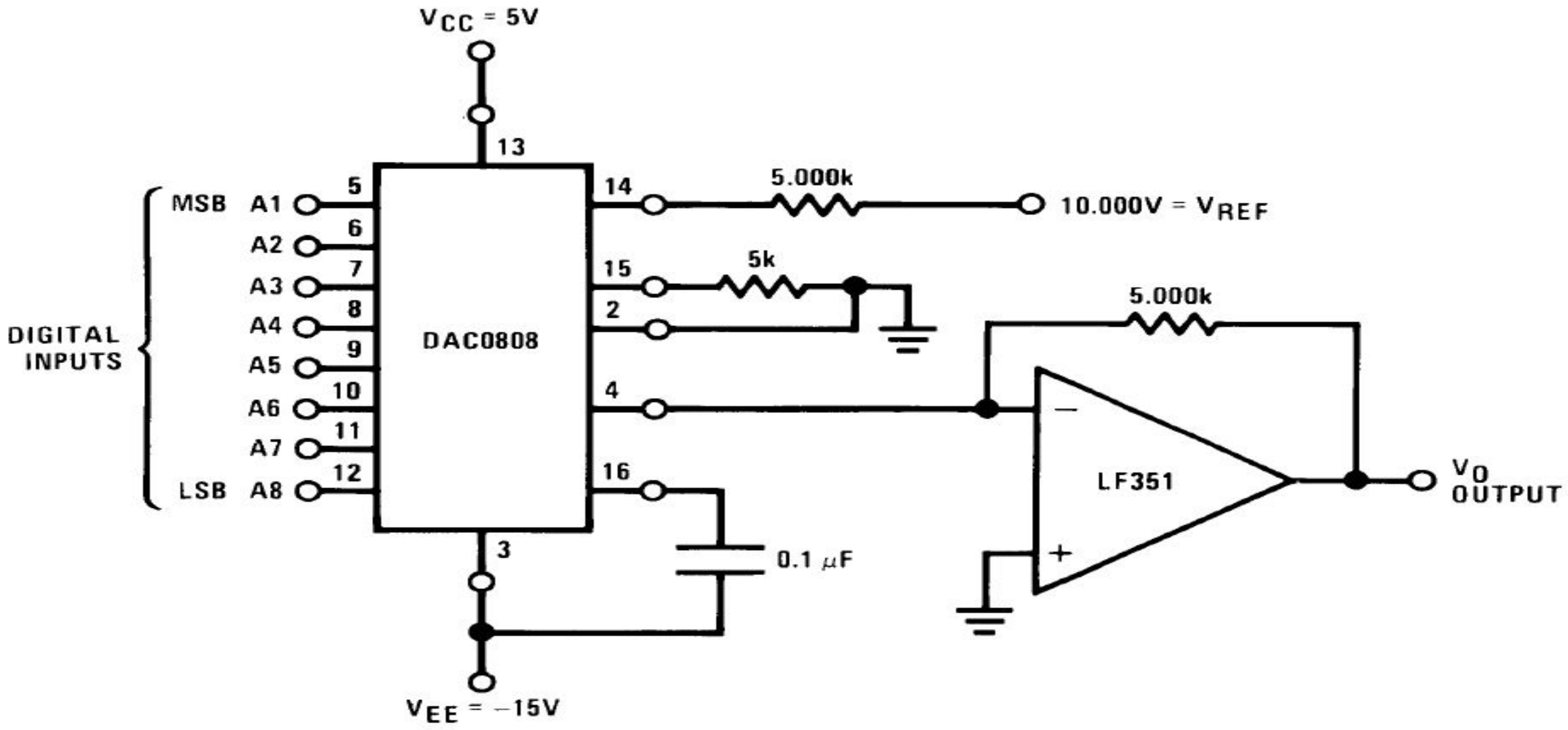
$$V_o = V_{ref} \times \left(\frac{A_1}{2} + \frac{A_2}{4} + \frac{A_3}{8} + \frac{A_4}{16} + \frac{A_5}{32} + \frac{A_6}{64} + \frac{A_7}{128} + \frac{A_8}{256}\right)$$

A1 = MSB ; A8 =LSB

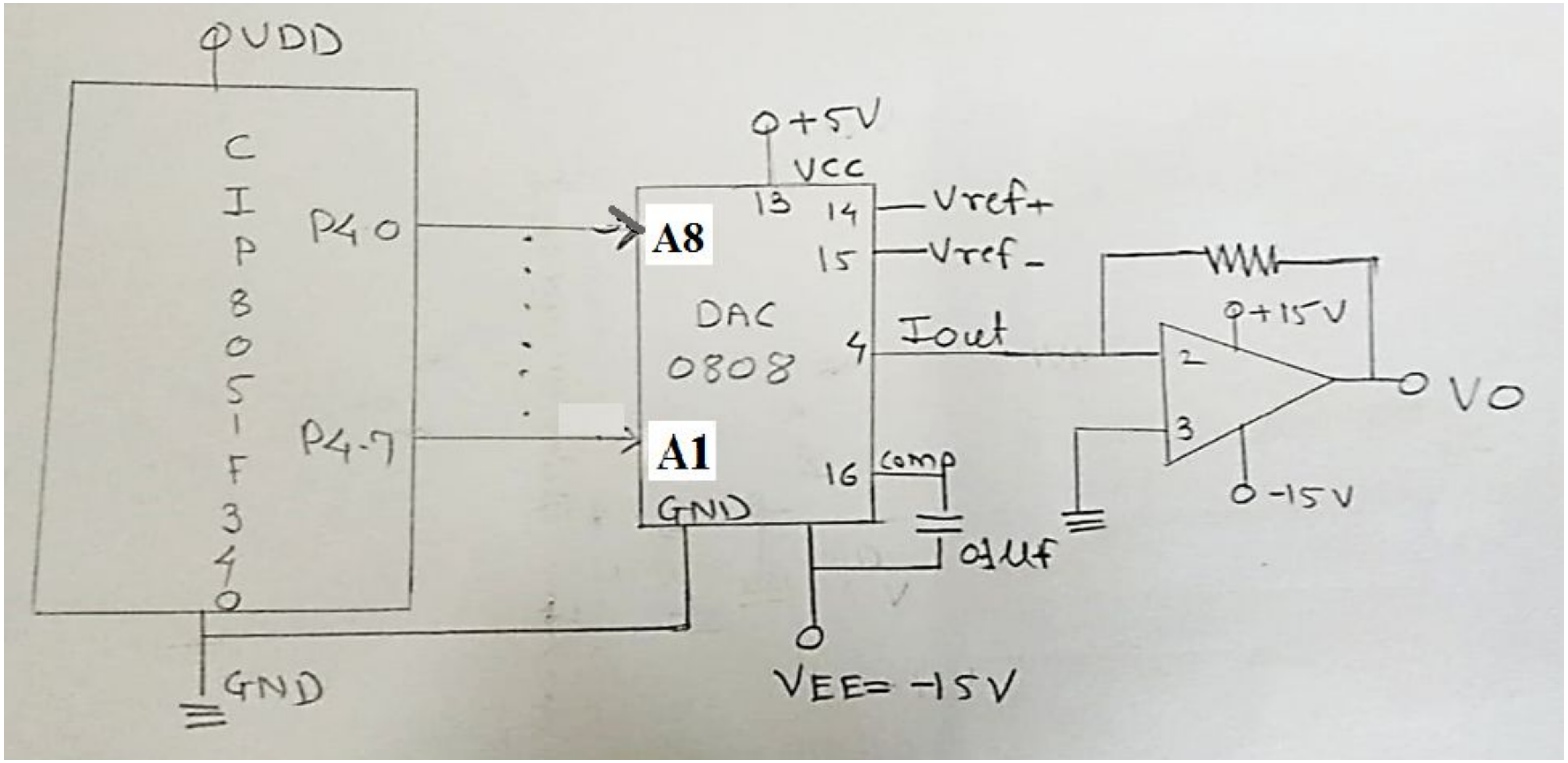
DAC 0808 Internal Structure



DAC Voltage Out



DAC Interfacing with C8051F340

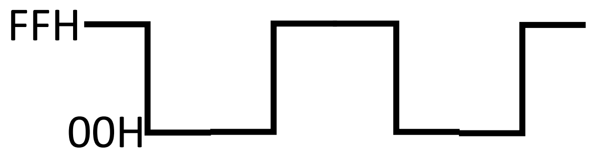


$$V_o = V_{ref} \times \left(\frac{A_1}{2} + \frac{A_2}{4} + \frac{A_3}{8} + \frac{A_4}{16} + \frac{A_5}{32} + \frac{A_6}{64} + \frac{A_7}{128} + \frac{A_8}{256}\right)$$

DAC PROGRAMING

- Square
- Triangular
- Sawtooth
- Sine

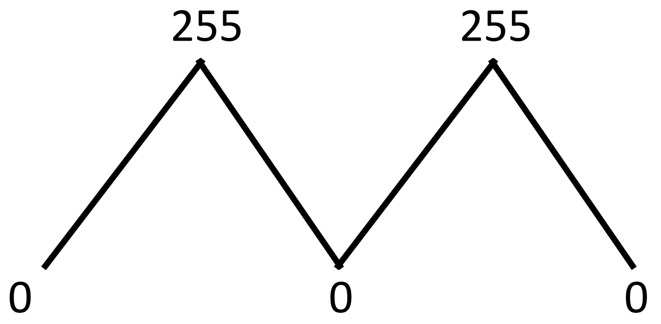
Square Wave



```
#include "c8051F340.h"
void DELAY_ms(unsigned int ms_Count)
{
   unsigned int i,j;
   for(i=0;i<ms_Count;i++)
   {
      for(j=0;j<100;j++);
   }
}
```

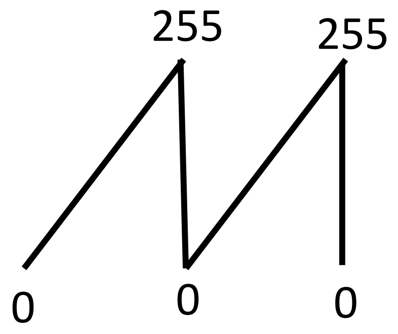
```
int main()
{
      P4MDOUT = 0xFF;     // Configure P4 as Digital Output
            P4= 0x00;
      while (1)
      {
                  P4 = ~P4;
                  DELAY_ms(0.5);
            }
            return 0;
}
```

Triangular



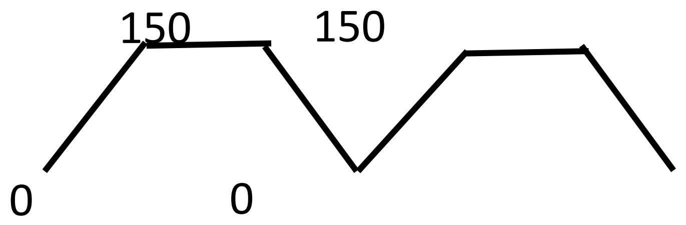
```
#include "c8051F340.h"
int main()
{
       int i;
      P4MDOUT = 0xFF; // All data lines Output
            while (1)
      {
                  for (i = 0; i <= 255; i++)
                  {
                          P4 = i;
                         }

                         for (i = 255; i >= 0; i--)
                   {
                          P4 = i;
                   }
      }
}
```

Sawtooth Wave

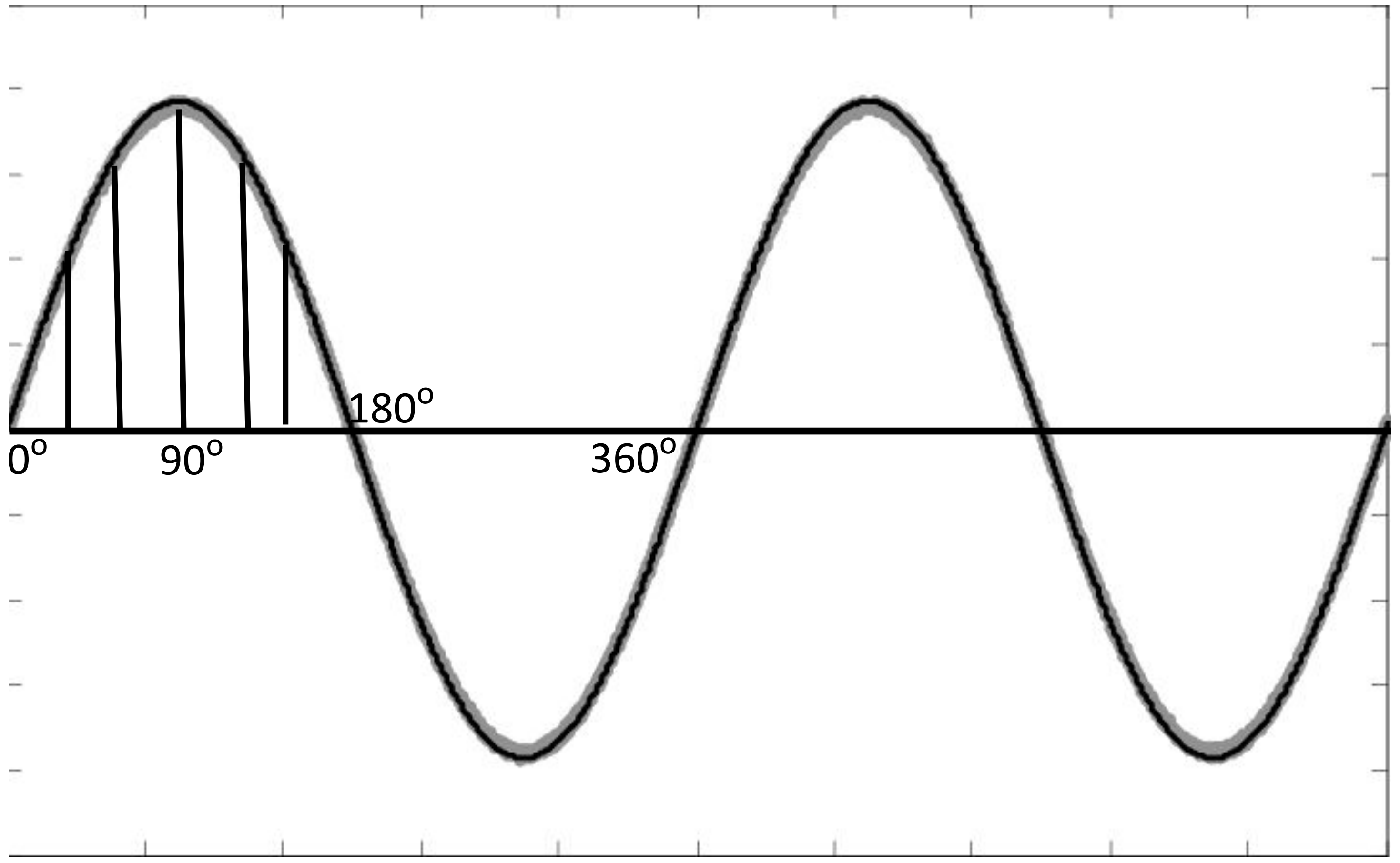


Complete program for sawtooth wave generation as home work

Trapezoidal Wave



Generating a Sine Wave



Generating a Sine Wave

- We first need a table whose values represent the magnitude of the sine of angles between 0 and 360 degrees
- The table values are integer numbers representing the voltage magnitude for the sine of theta
- This method ensures that only integer numbers are output to the DAC by the 8051 microcontroller
- Full-scale output of the DAC is achieved when all the data inputs of the DAC are high. Therefore, to achieve the full-scale 10 V output, we use the following equation:

$$V_{out} = 5\text{ V} + (5 \times \sin\theta)$$

- To find the value sent to the DAC for various angles, we simply multiply the V_{out} voltage by 25.60 because there are 256 steps and full-scale V_{out} is 10 volts. Therefore, 256 steps /10 V = 25.6 steps per volt.

Angle vs. Voltage Magnitude for Sine Wave

i	Angle (ɵ)	Sine ɵ	Voltage out Vo = 5+(5*Sine ɵ)	DAC count = Vo * 25.6
0	0	0	5	128
1	30	0.5	7.5	192
2	60	0.866	9.33	238
3	90	1.0	10	255
4	120	0.866	9.33	238
5	150	0.5	7.5	192
6	180	0	5	128
7	210	-0.5	2.5	64
8	240	-0.866	0.669	17
9	270	-1.0	0	0
10	300	-0.866	0.669	17
11	330	-0.5	2.5	64
12	360	0	5	128

Sine wave generation using DAC

```
#include "c8051F340.h"
unsigned char x[16] = {128,192,238,255,238,192,128,64,17,0,17,64};
int i;
int main(void)
{
          P4MDOUT = 0xFF;
           while(1)
          {
         for(i=0;i<12;i++)
         {
                      P4=x[i];
                 }
         }
 }
```

Note: You should use more samples i.e. higher value of i so that you will get smoother wave.

Angle vs. Voltage Magnitude for Sine Wave

i	Angle (ɵ)	Sine ɵ	Voltage out Vo = 5+(5*Sine ɵ)	DAC count = Vo * 25.6
	15	0.25	6.25	160
	45	0.707	8.53	219
	75	0.96	9.8	252
	105	0.965	9.825	252
	135	0.707	8.53	219
	165	0.25	6.25	160
	195	-0.25	3.7	95
	225	-0.707	1.46	37
	255	-0.965	0.17	4
	285	-0.965	0.17	4
	315	-0.707	1.46	37
	345	-0.25	3.7	95

Converting Digital Input to Analog Output

- Equation for converting Digital Input into its equivalent Analog output is,

$$V_o = V_{ref} \times (\frac{A_1}{2} + \frac{A_2}{4} + \frac{A_3}{8} + \frac{A_4}{16} + \frac{A_5}{32} + \frac{A_6}{64} + \frac{A_7}{128} + \frac{A_8}{256})$$

- Where, A1 is the MSB and A8 is the LSB

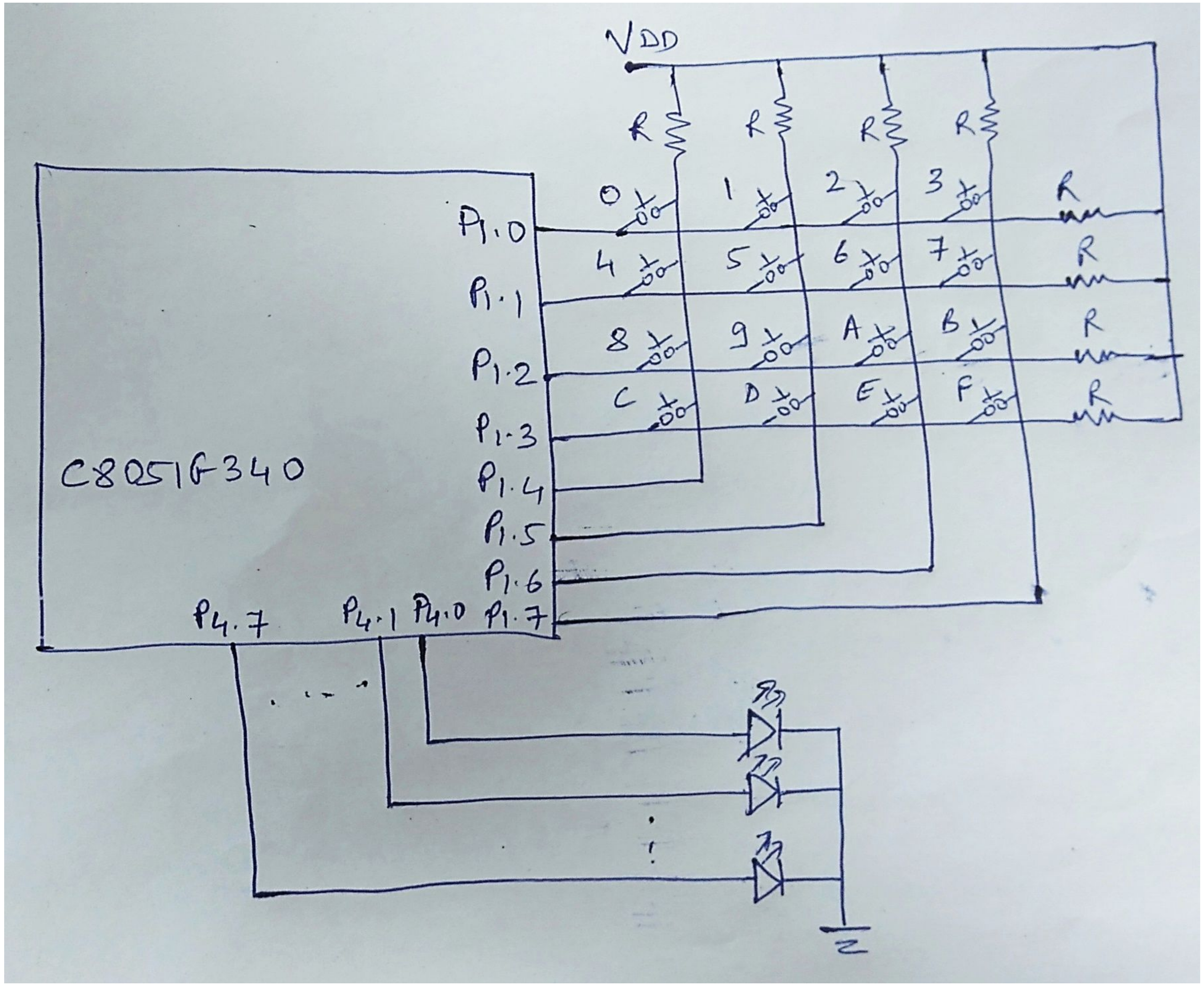
Considering Vref = 5V, calculate the Analog output obtained from the following Digital input:

Digital input	Analog Output
11001100	
11101101	
1000101	
11111	
10110110	

Considering Vref = 5V, calculate the Analog output obtained from the following Digital input:

Digital input	Analog Output
11001100	3.984V
11101101	4.6284V
1000101	2.69v
11111	0.605V
10110110	3.568V

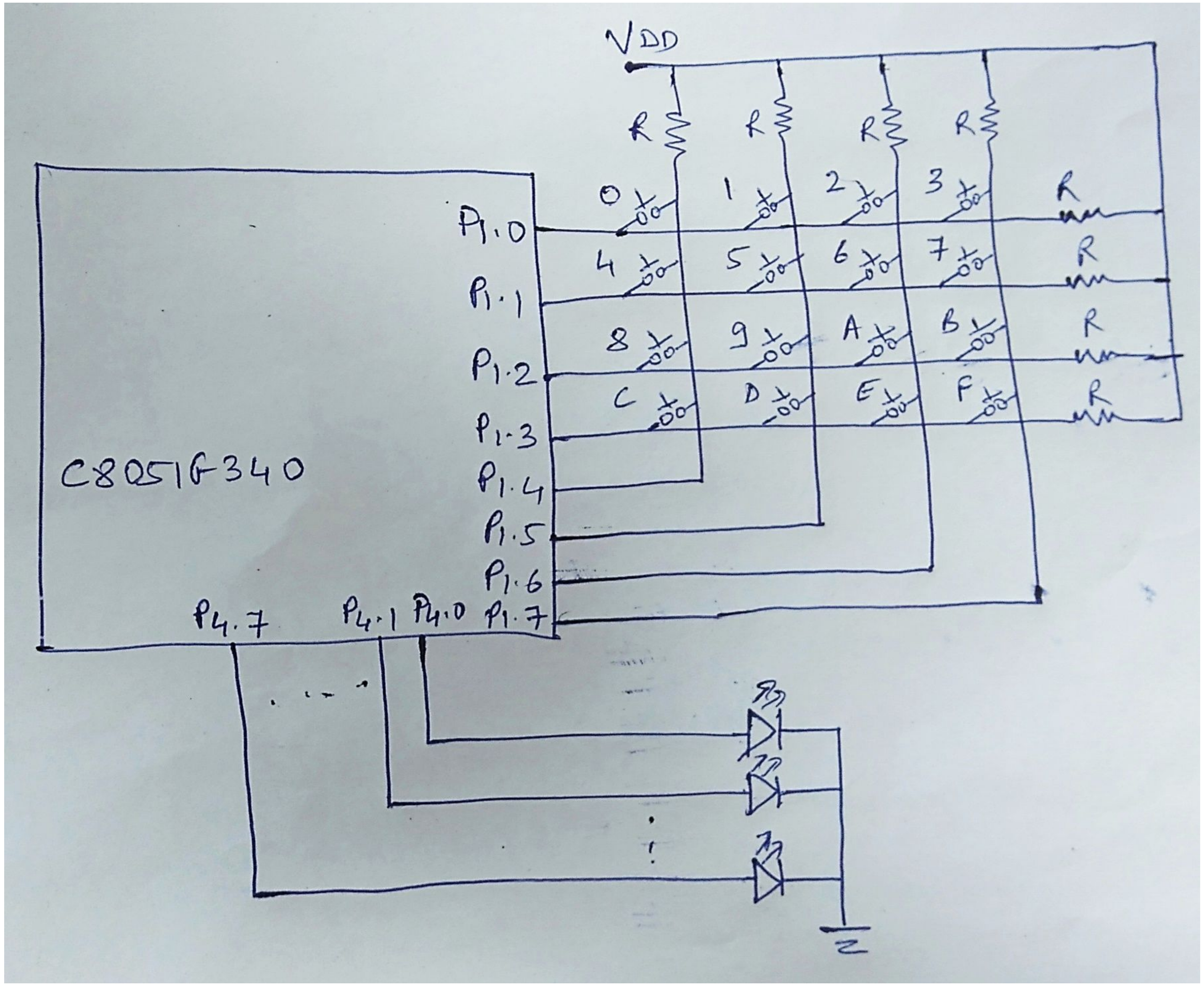
4x4 Matrix Keypad Interfacing diagram



Method to detect a pressed key - Grounding Rows and Reading Columns

- For detecting a key press,
  - Starting with the top row, the microcontroller grounds it by providing a low to row D0 only
    - It reads the columns, if the data read is all 1s, no key in that row is activated
      - The process is moved to the next row
  - It grounds the next row, reads the columns, and checks for any zero
    - Upon finding the row that the key press belongs to, it sets up the starting address for the look-up table holding the scan codes (or ASCII) for that row
  - To identify the key press, it rotates the column bits, one bit at a time, into the carry flag and checks to see if it is low
    - Upon finding the zero, it pulls out the ASCII code for that key from the look-up table
    - Otherwise, it increments the pointer to point to the next element of the look-up table

Keypad Interfacing diagram



Program

```
#include<C8051F340.h>
sbit  R1 = P1^0
sbit  R2 = P1^1
sbit  R3 = P1^2
sbit  R4 = P1^3
sbit  C1 = P1^4
sbit  C2 = P1^5
sbit  C3 = P1^6
sbit  C4 = P1^7
void msdelay (unsigned int);
```

Port pin definition

```
void main()
{     XBR1    = 0x40;                         // Enable Crossbar
      P1MDOUT = 0xF0;         //column output
      P1MDIN = 0x0F;          //rows input
           P4MDOUT = 0xFF;         //Display on port4
      while(1)
       {
      c1=0;                     // ground all column
      c2=0;
      c3=0;
      c4=0;
      while(R1==1&&R2==1&&R3==1&&R4==1); /* Scan Key for all keys released */
if(R1==0)
    {
        P1MDOUT = 0x0F;     //rows output
    P1MDIN = 0xF0;          //column  Input
        R1=0;               //Row0 grounded
        if(C1==0)
        {
            P4=0x00;
            msdelay(100);
        }
        else if(C2==0)
        {
            P4=0x01;
            msdelay(100);
        }
        else if(C3==0)
        {
            P4=0x02;
            msdelay(100);
        }
        else if(C4==0)
        {
            P4=0x03;
            msdelay(100);
        }
      }
    }
}
```

Detect row and Column

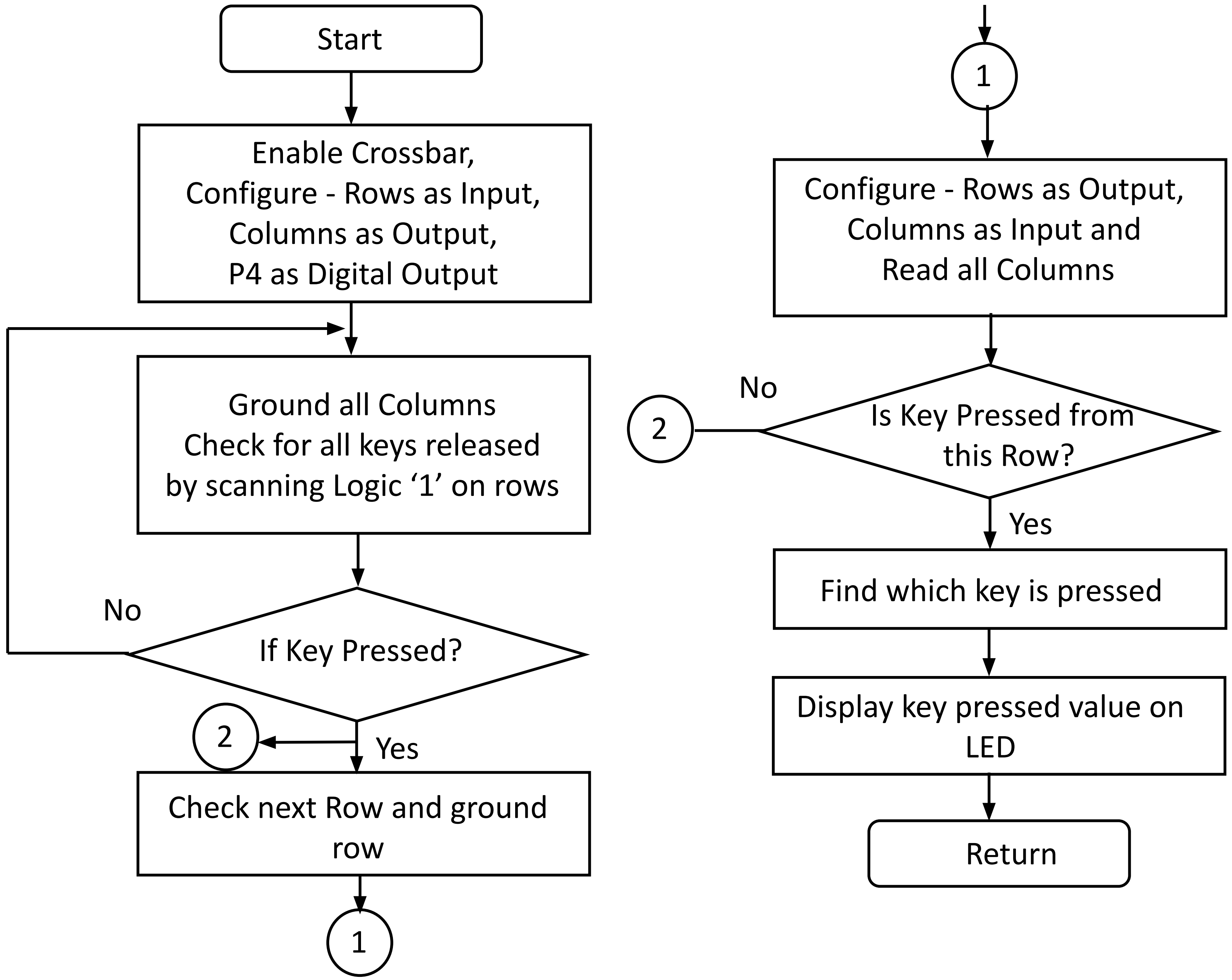
Display respective key

Here shown for Row 1 and Column 1 to 4

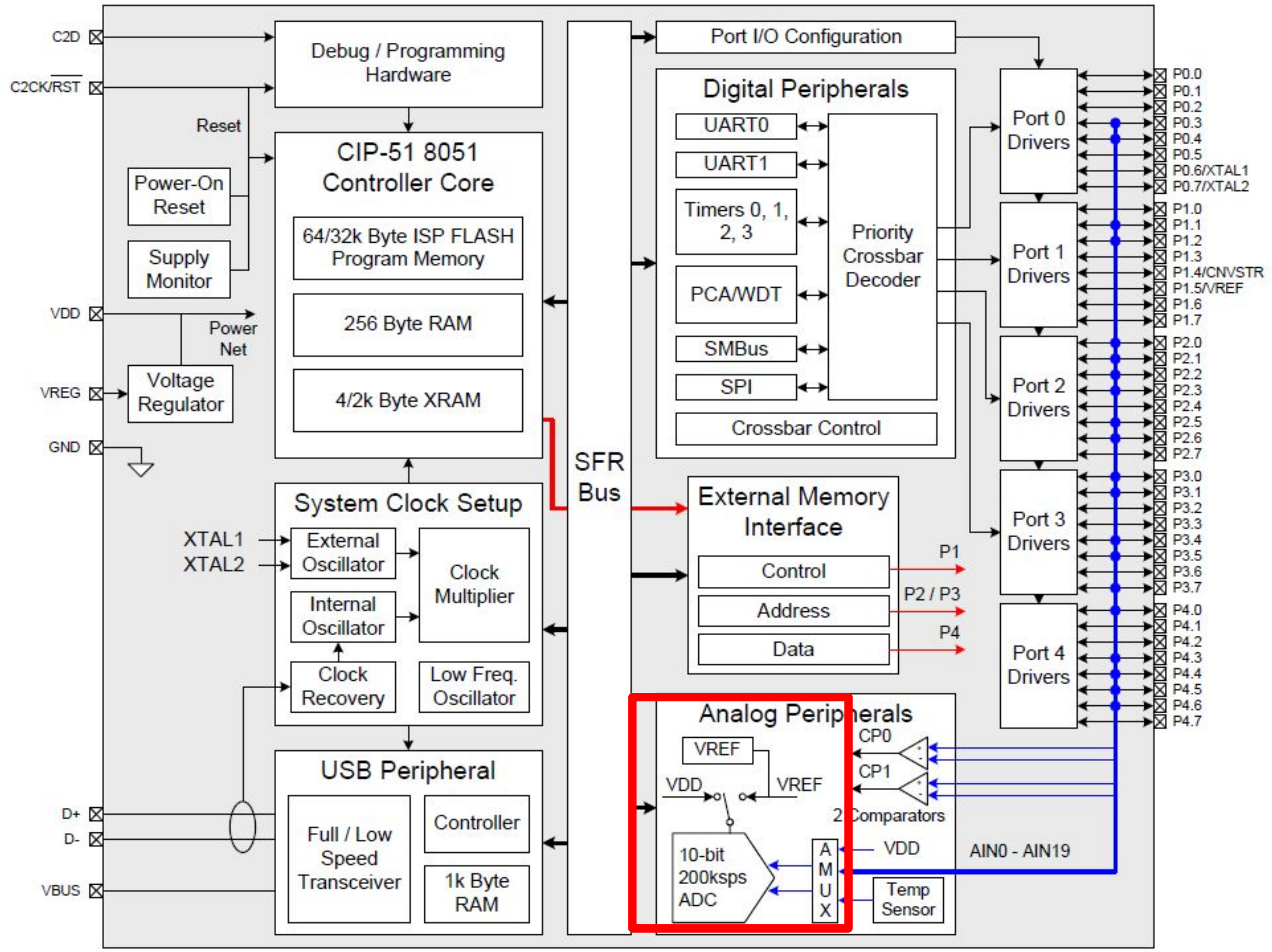
Same will repeat for all rows and columns

```
void msdelay (unsigned int itime)
  {
     unsigned int i, n;
     for (n=0; n<itime ; n++)
     {
          for (i=0; i<1535; i++); /* for 1 ms */
     }
  }
```

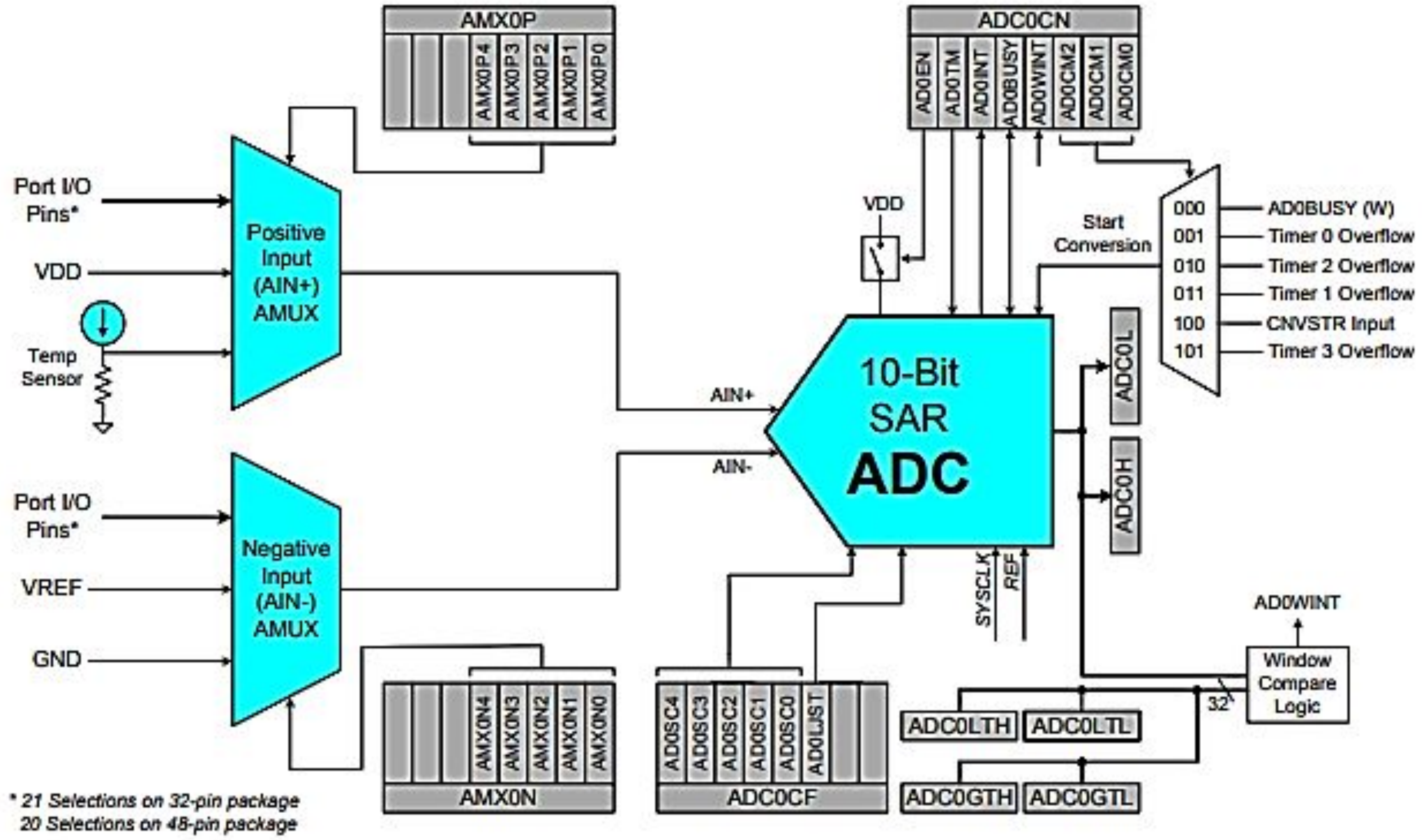
Flowchart


Start
Enable Crossbar, Configure - Rows as Input, Columns as Output, P4 as Digital Output
Ground all Columns Check for all keys released by scanning Logic '1' on rows
If Key Pressed?
No
Yes
2
Check next Row and ground row
1
1
Configure - Rows as Output, Columns as Input and Read all Columns
Is Key Pressed from this Row?
No
2
Yes
Find which key is pressed
Display key pressed value on LED
Return

Programming on chip ADC of C8051F340



10-Bit ADC (ADC0, C8051F340/1/2/3/4/5/6/7/A/B Only)



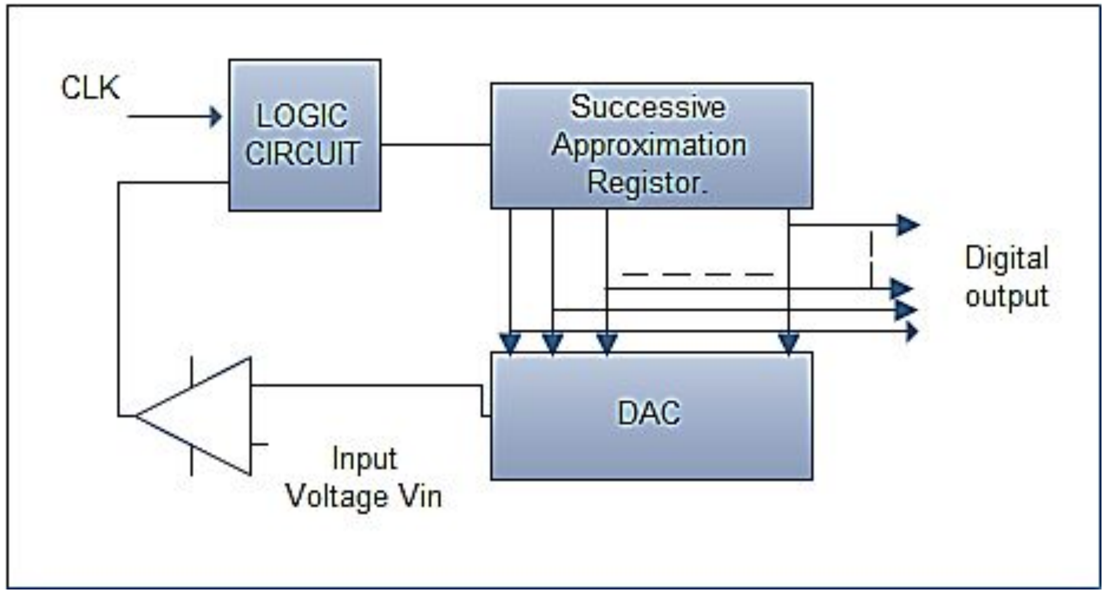
ADC0 of C8051F340- 10-Bit ADC

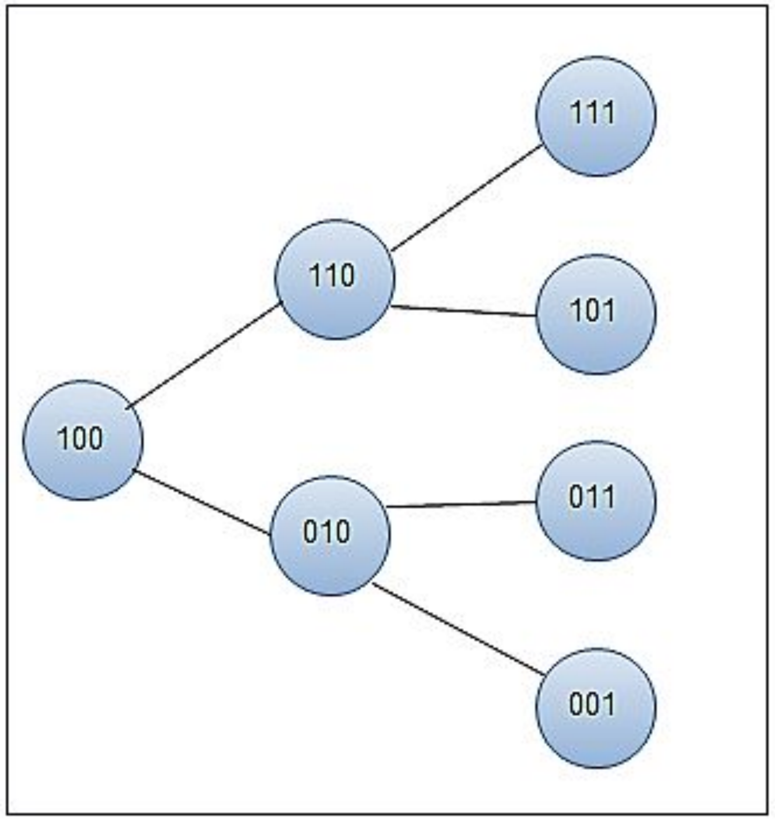
- The ADC0 subsystem for the C8051F34x devices consists of
  - Two analog multiplexers (referred to collectively as AMUX0),
  - 10-bit successive-approximation-register ADC with integrated track-and-hold and programmable window detector (200 kbps -maximum conversion speed)
- ADC0 operates in
  1. Single-ended &
  2. Differential modes
- ADC0 may be configured to measure
  1. Voltages at port pins,
  2. Temperature Sensor output, or
  3. VDD with respect to a port pin, VREF, or GND.
- The ADC0 subsystem is enabled only when the AD0EN bit in the ADC0 Control register (ADC0CN) is set to logic 1.
- The ADC0 subsystem is in low power shutdown when this bit is logic 0.

Analog Multiplexer

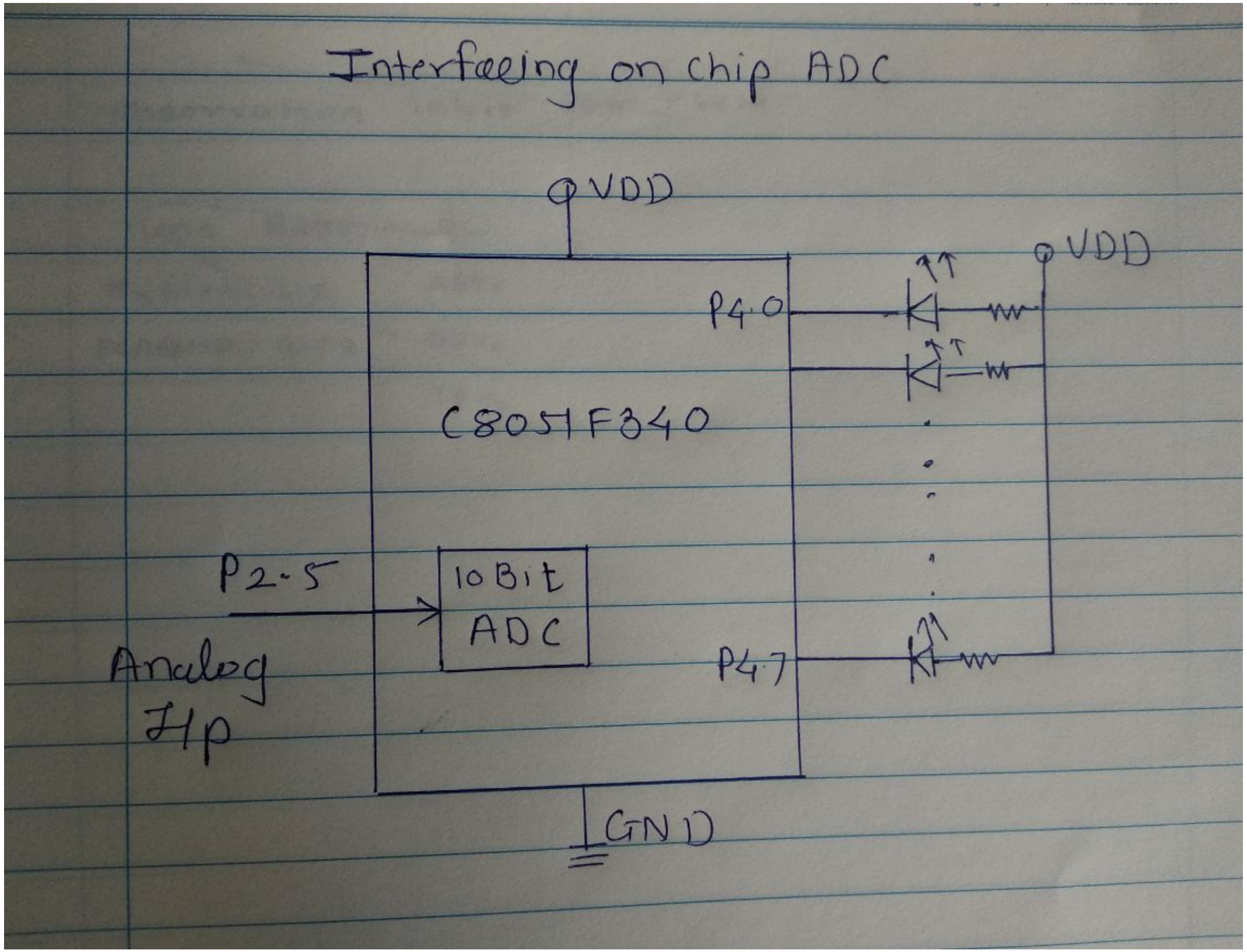
- AMUX0 selects the positive and negative inputs to the ADC.
- The positive input (AIN+) can be connected to individual Port pins, the on-chip temperature sensor, or the positive power supply (VDD).
- The negative input (AIN-) can be connected to individual Port pins, VREF, or GND.
- When GND is selected as the negative input, ADC0 operates in Single-ended Mode; at all other times, ADC0 operates in Differential Mode.

SAR Type ADC

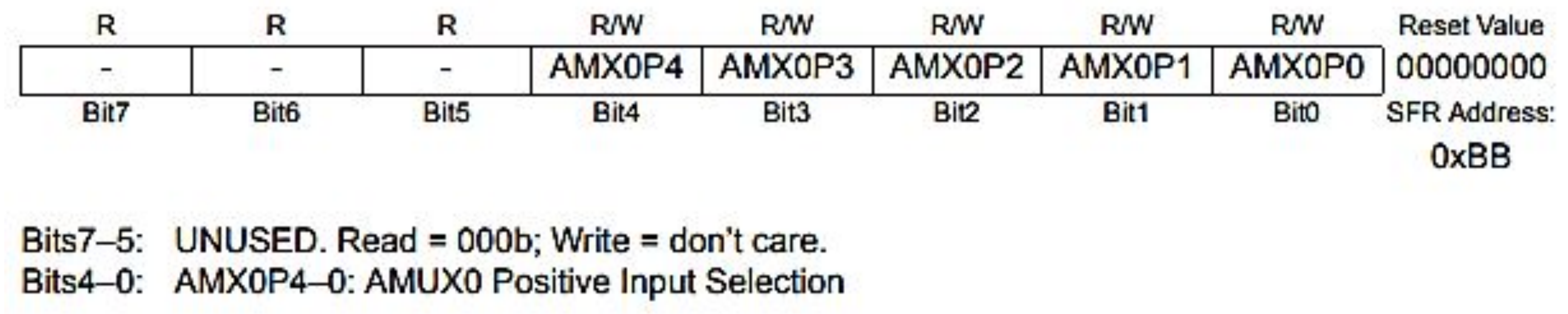

CLK
LOGIC CIRCUIT
Successive Approximation Registor.
Digital output
DAC
Input Voltage Vin


111
110
101
100
011
010
001

Interfacing Diagram for on-chip ADC programming



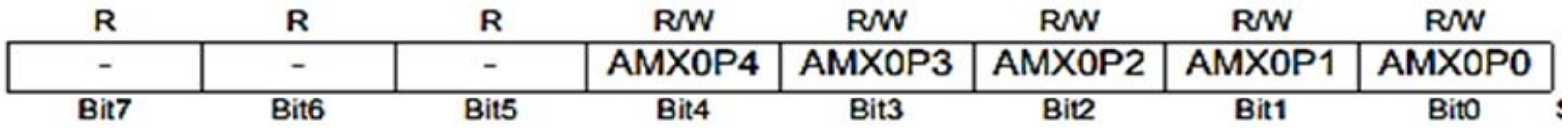
AMX0P: AMUX0 Positive Channel Select



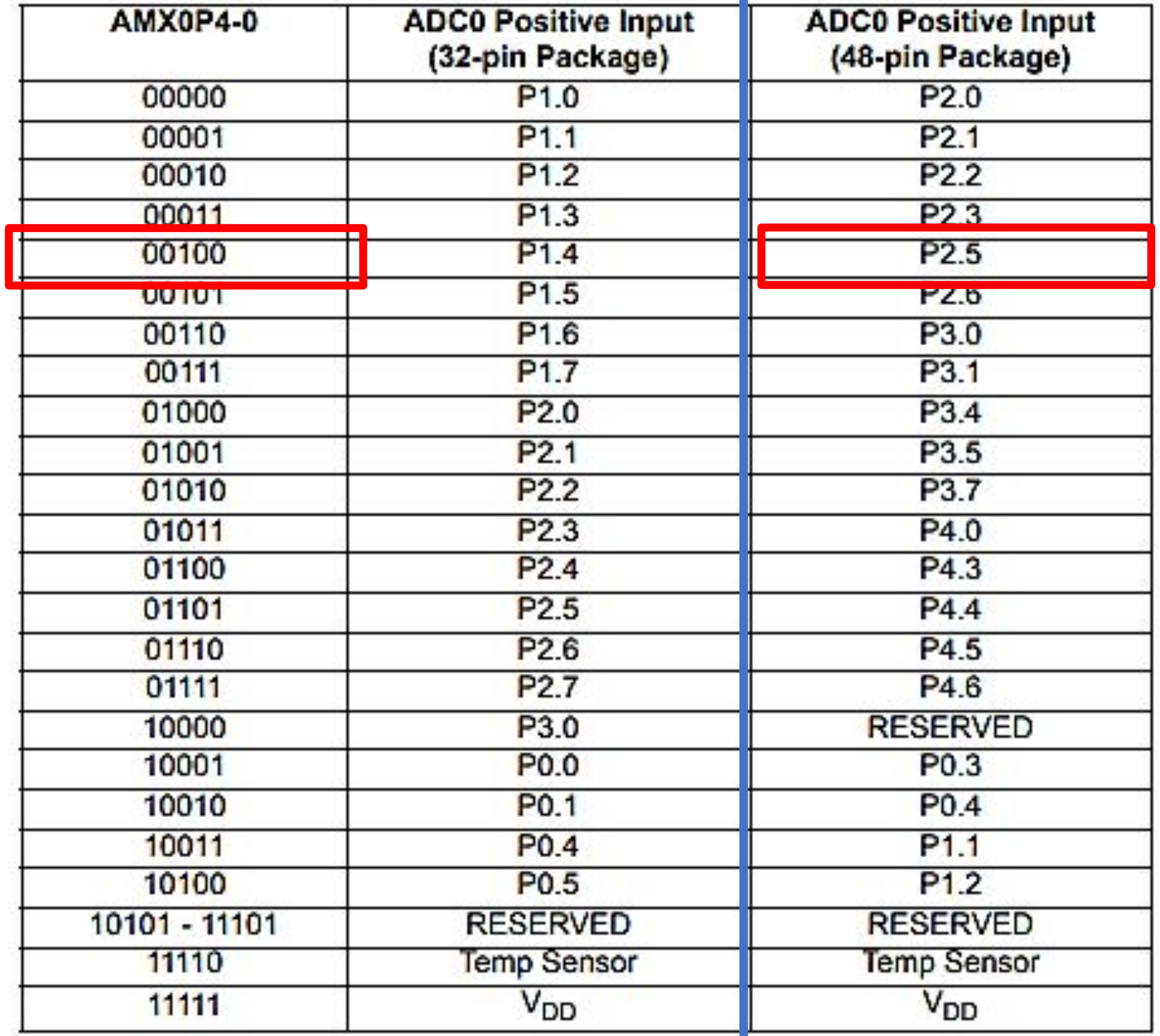
R	R	R	R/W	R/W	R/W	R/W	R/W	Reset Value
-	-	-	AMX0P4	AMX0P3	AMX0P2	AMX0P1	AMX0P0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xBB

Bits7–5: UNUSED. Read = 000b; Write = don't care.
Bits4–0: AMX0P4–0: AMUX0 Positive Input Selection

AMX0P:



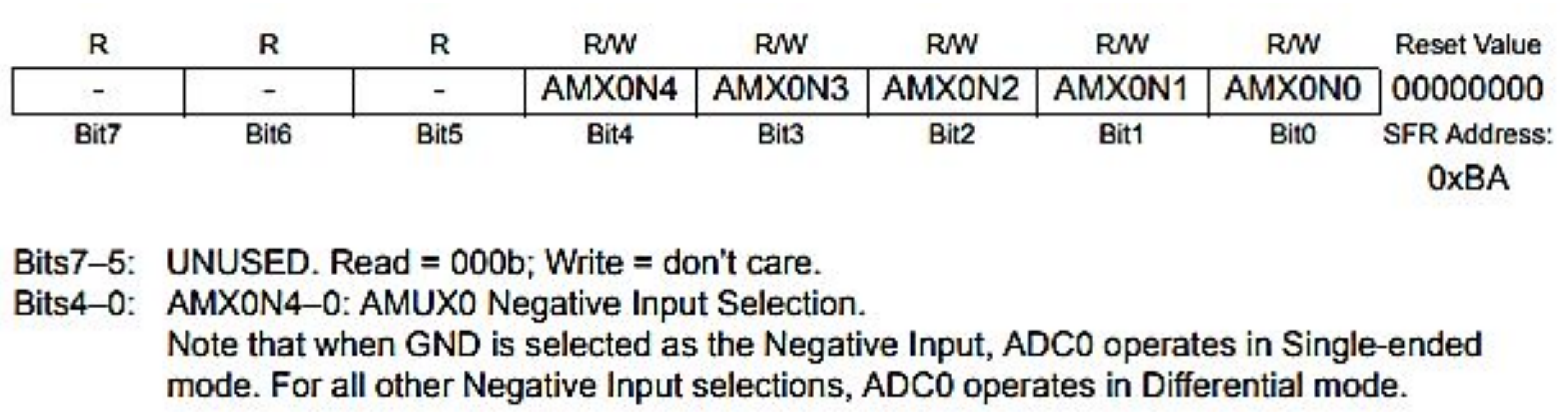
R	R	R	R/W	R/W	R/W	R/W	R/W
-	-	-	AMX0P4	AMX0P3	AMX0P2	AMX0P1	AMX0P0
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0



AMX0P4-0	ADC0 Positive Input (32-pin Package)	ADC0 Positive Input (48-pin Package)
00000	P1.0	P2.0
00001	P1.1	P2.1
00010	P1.2	P2.2
00011	P1.3	P2.3
00100	P1.4	P2.5
00101	P1.5	P2.6
00110	P1.6	P3.0
00111	P1.7	P3.1
01000	P2.0	P3.4
01001	P2.1	P3.5
01010	P2.2	P3.7
01011	P2.3	P4.0
01100	P2.4	P4.3
01101	P2.5	P4.4
01110	P2.6	P4.5
01111	P2.7	P4.6
10000	P3.0	RESERVED
10001	P0.0	P0.3
10010	P0.1	P0.4
10011	P0.4	P1.1
10100	P0.5	P1.2
10101 - 11101	RESERVED	RESERVED
11110	Temp Sensor	Temp Sensor
11111	V_{DD}	V_{DD}

AMX0P = ____ ;   // P2.5 as ANIP+

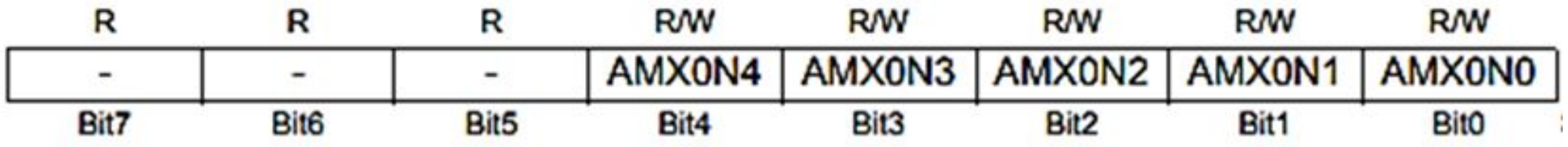
AMX0N: AMUX0 Negative Channel Select



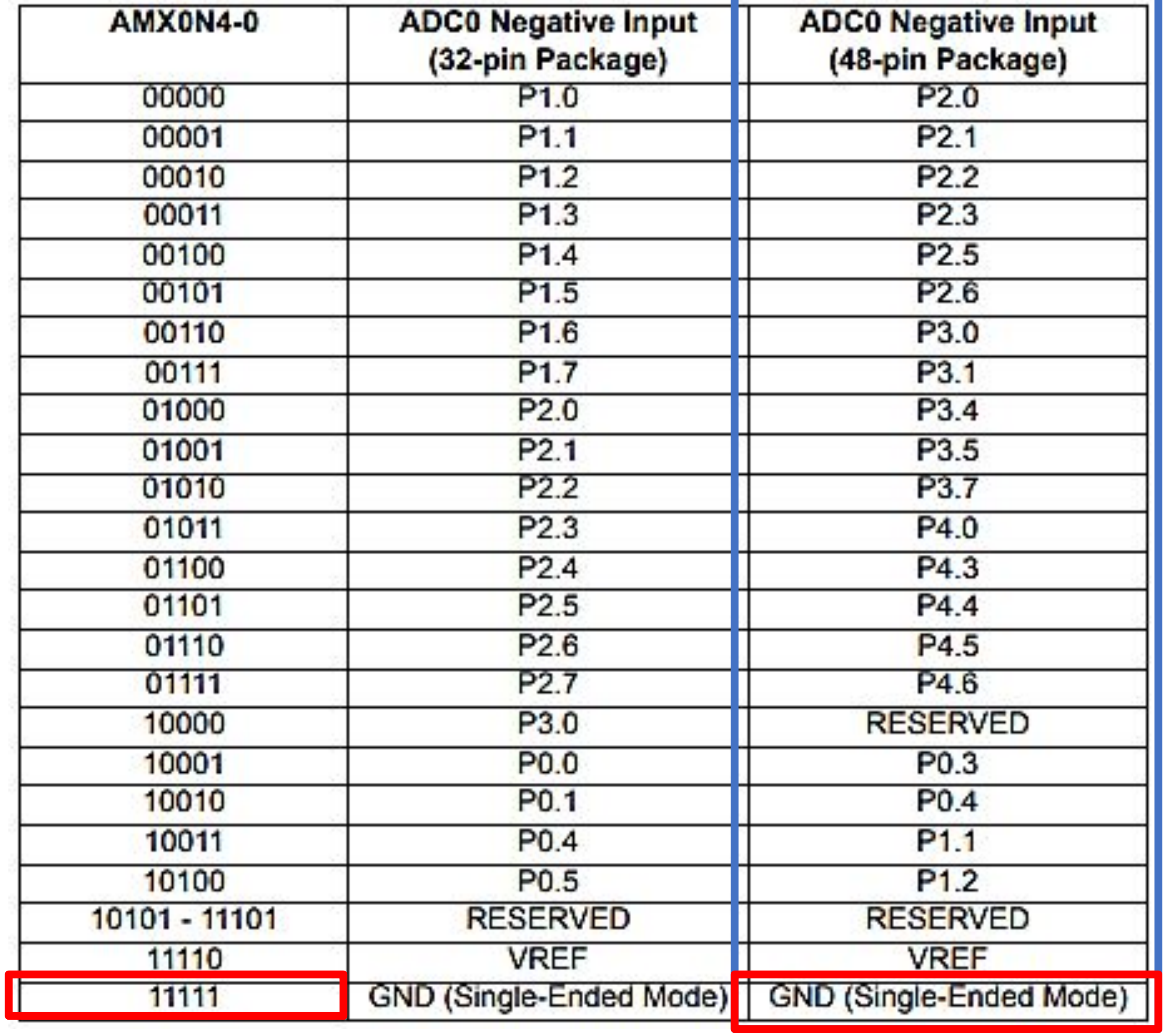
R	R	R	R/W	R/W	R/W	R/W	R/W	Reset Value
-	-	-	AMX0N4	AMX0N3	AMX0N2	AMX0N1	AMX0N0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xBA

Bits7–5: UNUSED. Read = 000b; Write = don't care.
Bits4–0: AMX0N4–0: AMUX0 Negative Input Selection.
Note that when GND is selected as the Negative Input, ADC0 operates in Single-ended mode. For all other Negative Input selections, ADC0 operates in Differential mode.

AMX0N:



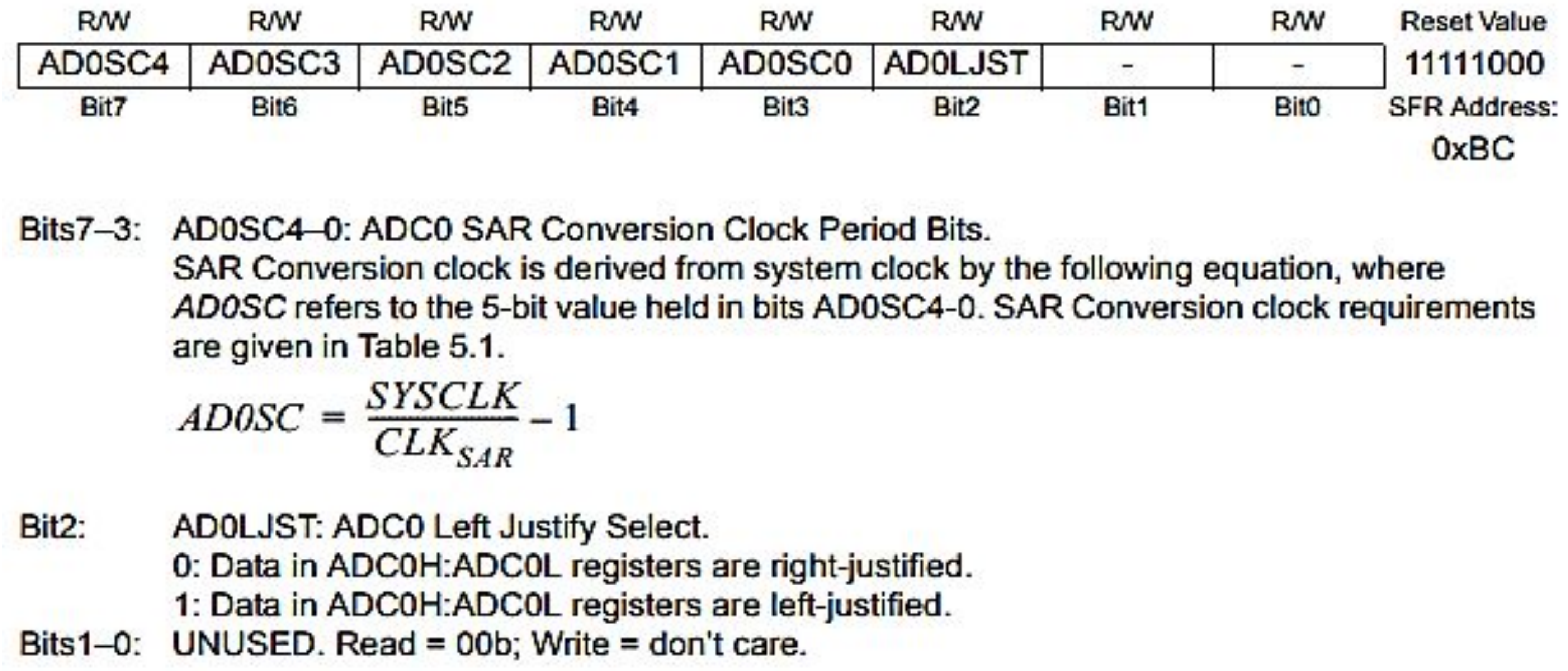
R	R	R	R/W	R/W	R/W	R/W	R/W
-	-	-	AMX0N4	AMX0N3	AMX0N2	AMX0N1	AMX0N0
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0



AMX0N4-0	ADC0 Negative Input (32-pin Package)	ADC0 Negative Input (48-pin Package)
00000	P1.0	P2.0
00001	P1.1	P2.1
00010	P1.2	P2.2
00011	P1.3	P2.3
00100	P1.4	P2.5
00101	P1.5	P2.6
00110	P1.6	P3.0
00111	P1.7	P3.1
01000	P2.0	P3.4
01001	P2.1	P3.5
01010	P2.2	P3.7
01011	P2.3	P4.0
01100	P2.4	P4.3
01101	P2.5	P4.4
01110	P2.6	P4.5
01111	P2.7	P4.6
10000	P3.0	RESERVED
10001	P0.0	P0.3
10010	P0.1	P0.4
10011	P0.4	P1.1
10100	P0.5	P1.2
10101 - 11101	RESERVED	RESERVED
11110	VREF	VREF
11111	GND (Single-Ended Mode)	GND (Single-Ended Mode)

AMX0N = ____ ; //Connect ANIP- to GND for single ended ADC

ADC0CF: ADC0 Configuration



R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
AD0SC4	AD0SC3	AD0SC2	AD0SC1	AD0SC0	AD0LJST	-	-	11111000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xBC

Bits7–3: AD0SC4–0: ADC0 SAR Conversion Clock Period Bits.
SAR Conversion clock is derived from system clock by the following equation, where *AD0SC* refers to the 5-bit value held in bits AD0SC4-0. SAR Conversion clock requirements are given in Table 5.1.

$$AD0SC = \frac{SYSCLK}{CLK_{SAR}} - 1$$

Bit2: AD0LJST: ADC0 Left Justify Select.
0: Data in ADC0H:ADC0L registers are right-justified.
1: Data in ADC0H:ADC0L registers are left-justified.

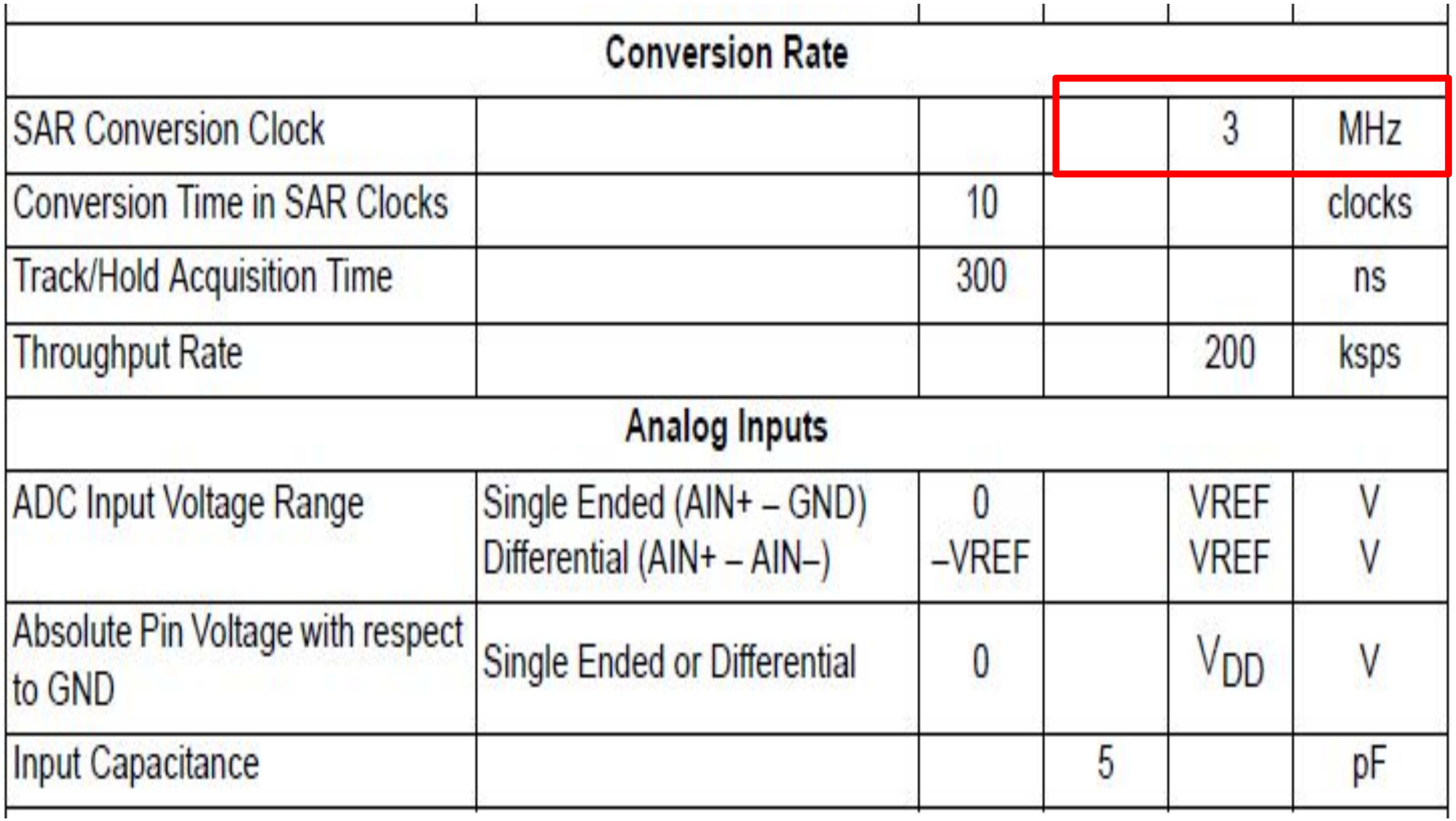
Bits1–0: UNUSED. Read = 00b; Write = don't care.

ADC0CF= (((SYSCLK/3000000)-1)<<3);   12MHz/3MHz-1=3

Right Justified = 0000 0011 1111 1111   **Left Justified = 1111 1111 1100 0000**

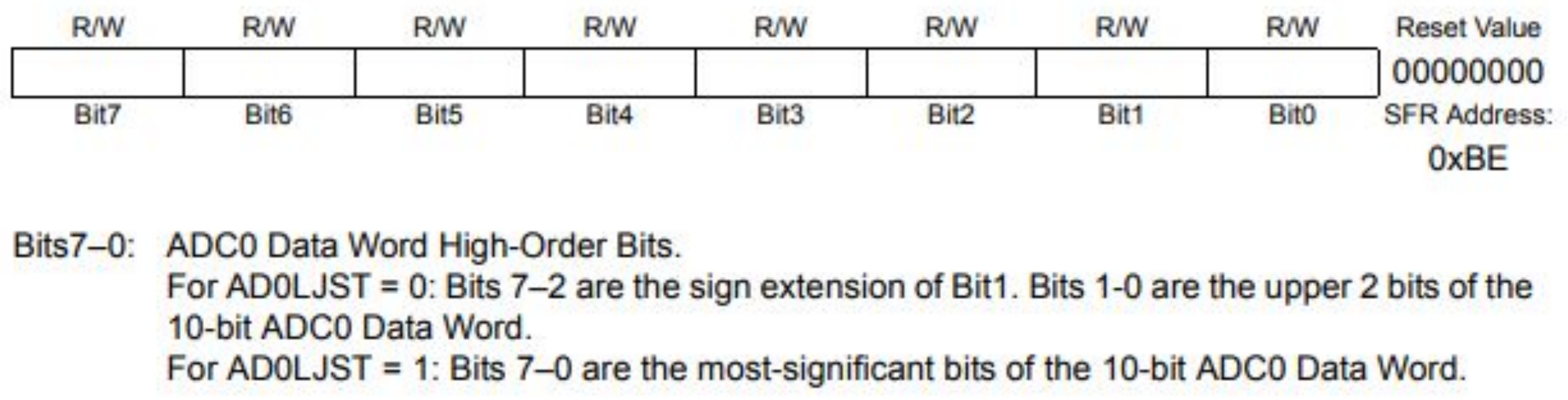
0 3 F F   **F F C 0**

Table 5.1. ADC0 Electrical Characteristics



Conversion Rate					
SAR Conversion Clock				3	MHz
Conversion Time in SAR Clocks		10			clocks
Track/Hold Acquisition Time		300			ns
Throughput Rate				200	ksps
Analog Inputs					
ADC Input Voltage Range	Single Ended (AIN+ – GND) Differential (AIN+ – AIN–)	0 –VREF		VREF VREF	V V
Absolute Pin Voltage with respect to GND	Single Ended or Differential	0		V_{DD}	V
Input Capacitance			5		pF

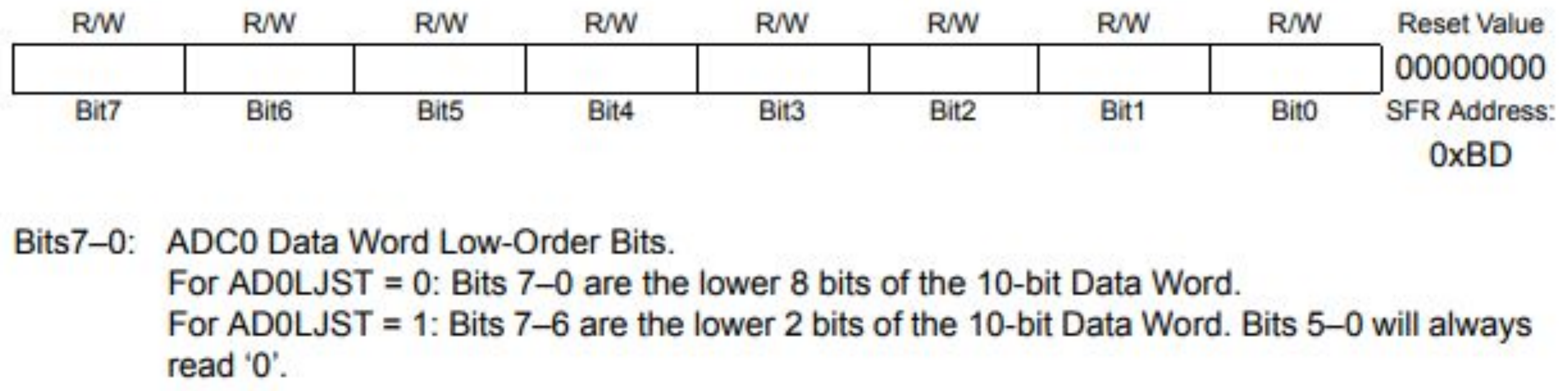
ADC0H: ADC0 Data Word MSB



R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xBE

Bits7–0: ADC0 Data Word High-Order Bits.
For AD0LJST = 0: Bits 7–2 are the sign extension of Bit1. Bits 1-0 are the upper 2 bits of the 10-bit ADC0 Data Word.
For AD0LJST = 1: Bits 7–0 are the most-significant bits of the 10-bit ADC0 Data Word.

ADC0L: ADC0 Data Word LSB

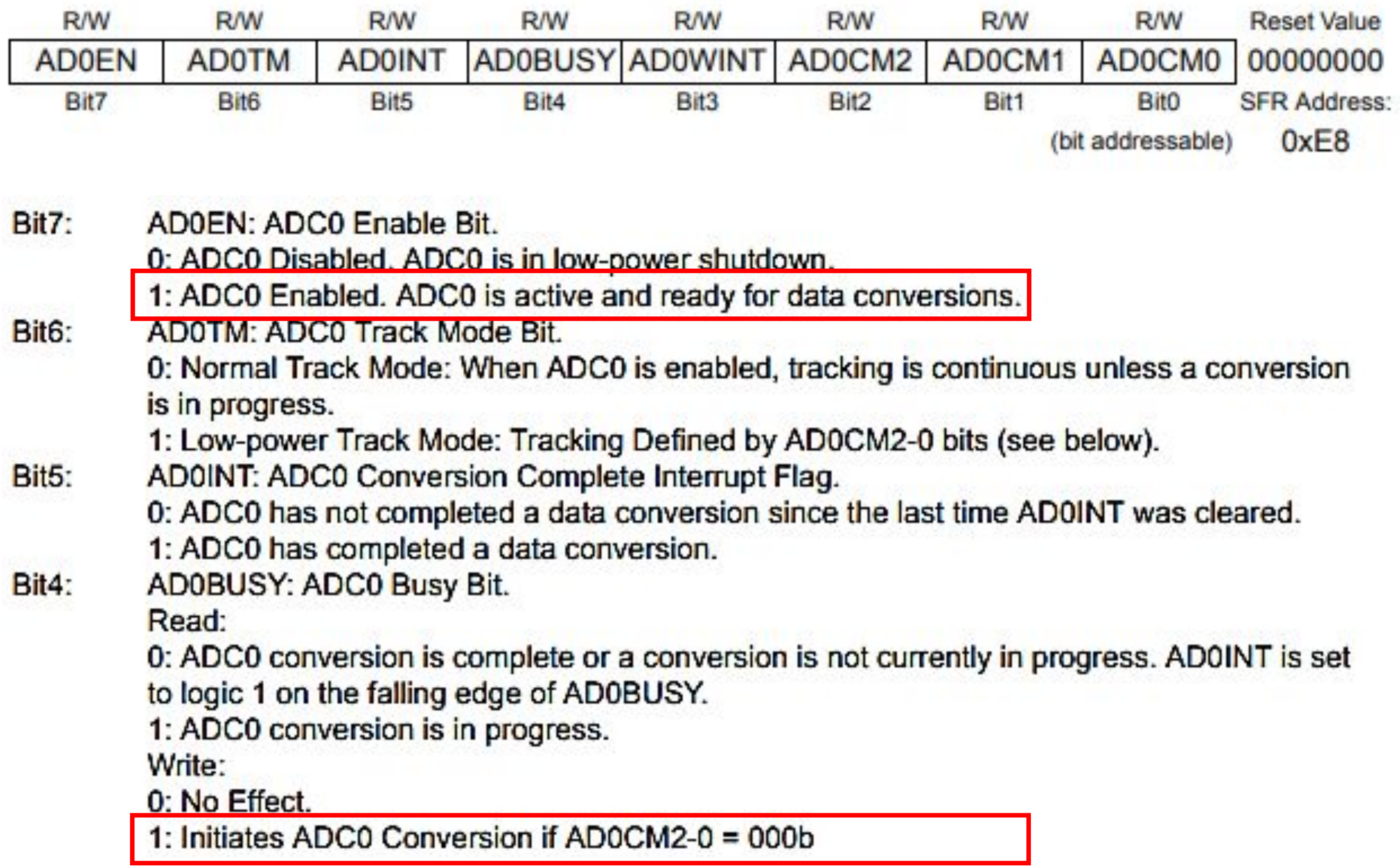


R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xBD

Bits7–0: ADC0 Data Word Low-Order Bits.
For AD0LJST = 0: Bits 7–0 are the lower 8 bits of the 10-bit Data Word.
For AD0LJST = 1: Bits 7–6 are the lower 2 bits of the 10-bit Data Word. Bits 5–0 will always read ‘0’.

ADC0CN: ADC0 Control

ADC0CN = ______;



R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
AD0EN	AD0TM	AD0INT	AD0BUSY	AD0WINT	AD0CM2	AD0CM1	AD0CM0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:
							(bit addressable)	0xE8

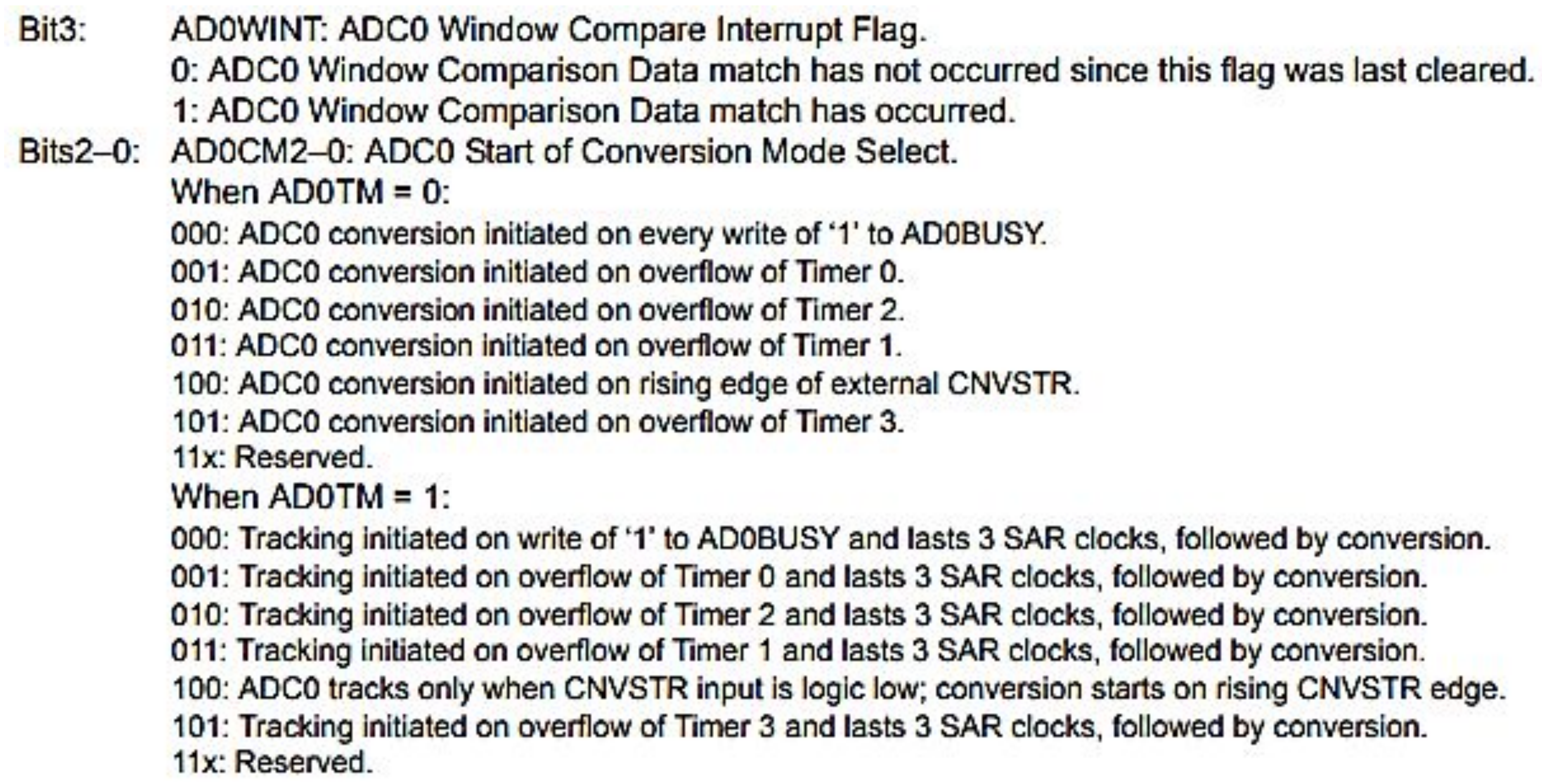
Bit7: AD0EN: ADC0 Enable Bit.
0: ADC0 Disabled. ADC0 is in low-power shutdown.
1: ADC0 Enabled. ADC0 is active and ready for data conversions.

Bit6: AD0TM: ADC0 Track Mode Bit.
0: Normal Track Mode: When ADC0 is enabled, tracking is continuous unless a conversion is in progress.
1: Low-power Track Mode: Tracking Defined by AD0CM2-0 bits (see below).

Bit5: AD0INT: ADC0 Conversion Complete Interrupt Flag.
0: ADC0 has not completed a data conversion since the last time AD0INT was cleared.
1: ADC0 has completed a data conversion.

Bit4: AD0BUSY: ADC0 Busy Bit.
Read:
0: ADC0 conversion is complete or a conversion is not currently in progress. AD0INT is set to logic 1 on the falling edge of AD0BUSY.
1: ADC0 conversion is in progress.
Write:
0: No Effect.
1: Initiates ADC0 Conversion if AD0CM2-0 = 000b

ADC0CN: ADC0 Control



Bit3: AD0WINT: ADC0 Window Compare Interrupt Flag.
0: ADC0 Window Comparison Data match has not occurred since this flag was last cleared.
1: ADC0 Window Comparison Data match has occurred.

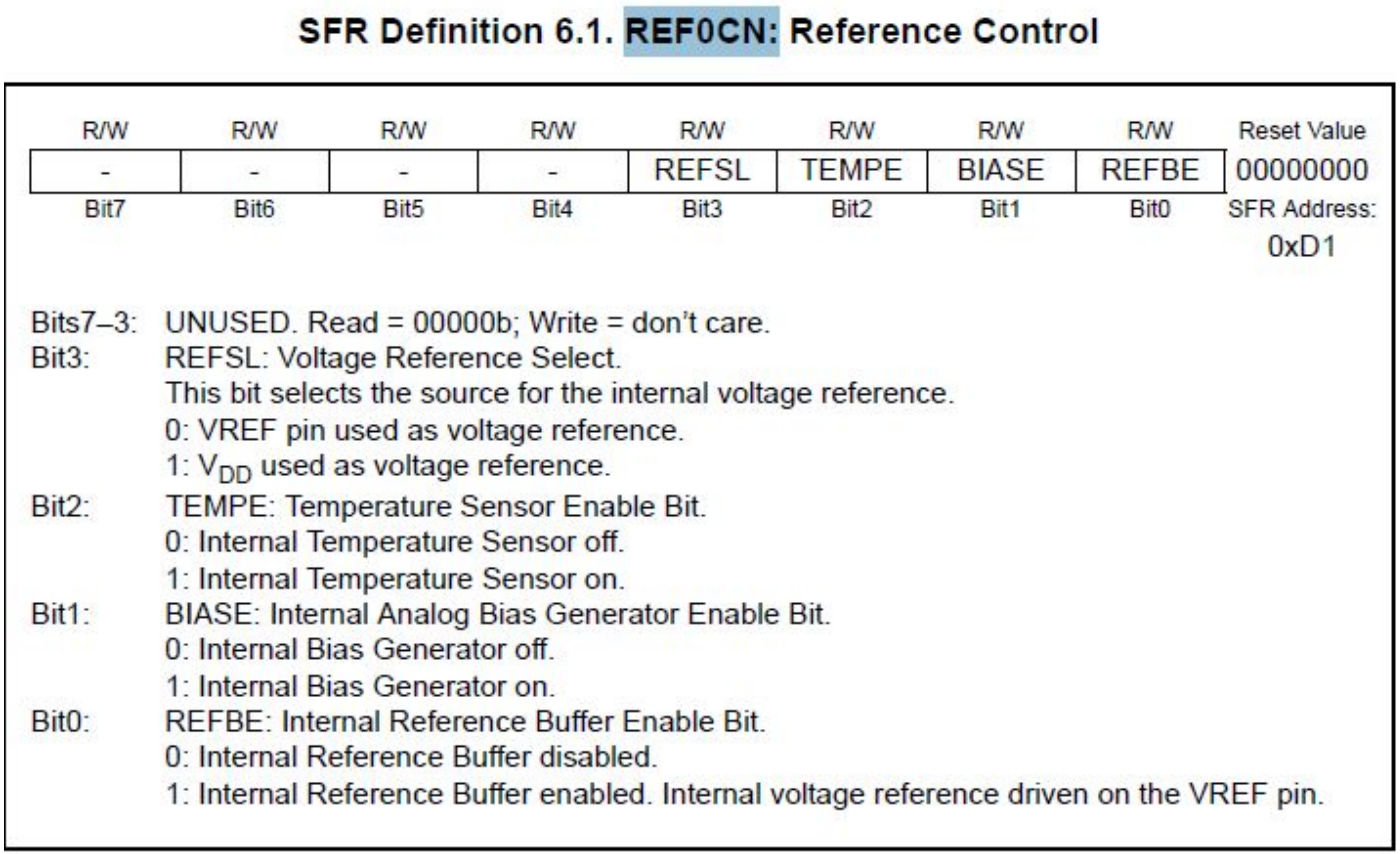
Bits2–0: AD0CM2–0: ADC0 Start of Conversion Mode Select.

When AD0TM = 0:
000: ADC0 conversion initiated on every write of '1' to AD0BUSY.
001: ADC0 conversion initiated on overflow of Timer 0.
010: ADC0 conversion initiated on overflow of Timer 2.
011: ADC0 conversion initiated on overflow of Timer 1.
100: ADC0 conversion initiated on rising edge of external CNVSTR.
101: ADC0 conversion initiated on overflow of Timer 3.
11x: Reserved.

When AD0TM = 1:
000: Tracking initiated on write of '1' to AD0BUSY and lasts 3 SAR clocks, followed by conversion.
001: Tracking initiated on overflow of Timer 0 and lasts 3 SAR clocks, followed by conversion.
010: Tracking initiated on overflow of Timer 2 and lasts 3 SAR clocks, followed by conversion.
011: Tracking initiated on overflow of Timer 1 and lasts 3 SAR clocks, followed by conversion.
100: ADC0 tracks only when CNVSTR input is logic low; conversion starts on rising CNVSTR edge.
101: Tracking initiated on overflow of Timer 3 and lasts 3 SAR clocks, followed by conversion.
11x: Reserved.

REF0CN: Reference Control

REF0CN = ______; // Set VDD as reference voltage for ADC



SFR Definition 6.1. REF0CN: Reference Control

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
-	-	-	-	REFSL	TEMPE	BIASE	REFBE	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xD1

Bits7–3:	UNUSED. Read = 00000b; Write = don't care.
Bit3:	REFSL: Voltage Reference Select. This bit selects the source for the internal voltage reference. 0: VREF pin used as voltage reference. 1: V_{DD} used as voltage reference.
Bit2:	TEMPE: Temperature Sensor Enable Bit. 0: Internal Temperature Sensor off. 1: Internal Temperature Sensor on.
Bit1:	BIASE: Internal Analog Bias Generator Enable Bit. 0: Internal Bias Generator off. 1: Internal Bias Generator on.
Bit0:	REFBE: Internal Reference Buffer Enable Bit. 0: Internal Reference Buffer disabled. 1: Internal Reference Buffer enabled. Internal voltage reference driven on the VREF pin.

Steps for ADC Programming

- **Configure ADC**
- **Enable ADC**
- **Give START of CONVERSION (SOC)**
- **Wait for the END of CONVERSION (EOC)**
- **Read the 10-bit A/D value from ADC0H and ADC0L registers**
- **Use it for further processing or just display on LED or UART.**

Steps for Configuring ADC

- Configure the GPIO pin for ADC function using PxMDIN or PxSKIP register P2SKIP=0X20 (P2.5 as analog Input) P2MDIN = 0XDF;
- Select P2.5 as analog input pin from AMX0P =0x04
- Select GND pin for single ended ADC of AMX0N =0x1F
- Configure ADC0CF for SAR Conversion clock is derived from system clock using formula:

  ADC0CF= (((SYSCLK/3000000)-1)<<3);

- CLK SAR =3MHz from table 5.1
- Configure VDD as reference voltage for ADC with the help of REF0CN =0X08
- Enable ADC(AD0EN)and give START OF CONVERSION (AD0CM2-ADCCM0=000 - ADCBUSY) using ADC0CN=0x90
- Wait for the END Of CONVERSION to complete, ADC0CR.AD0BUSY=0
- Read the 10-bit A/D value from ADC0H and ADC0L registers
- Use it for further processing or just display on LED or UART.

Program for ADC

- //Pin 2.5 as Analog input

```
#include "c8051F340.h"
#define SYSCLK 12000000
void main ()
{
OSCICN = 0X83;          //System Clock 12 MHz
XBR1 = 0X40;            // Enable crossbar
P4MDOUT = 0XFF;         // P4 LEDs connected make o/p
P2MDIN = 0XDF;          // P2.5  analog Input
P2SKIP = 0X20;          // Skip P2.5
REF0CN = 0X08;          // Set VDD as reference voltage for  ADC
AMX0P = 0X04;           // P2.5 as ANIP+
AMX0N = 0X1F;           //Connect ANIP- to GND for single ended ADC
ADC0CF = (((SYSCLK/3000000)-1)<<3);
AD0EN = 1;              // Enable ADC
{
ADC0CN = 0X90;          // Give SOC
while(AD0BUSY == 1); // Wait for  EOC
delay();
P4 =~ADC0L;             // Display 10-bit A/D value on LED
delay();
P4 =~ADC0H;
delay();
}
while(1);
}
```

Conversion of Analog Input to equivalent Digital Output

- Dout= (Vin / Vref)*1024
- Vin= (Dout*Vref)/1024
- Vref= VDD=3.3 V

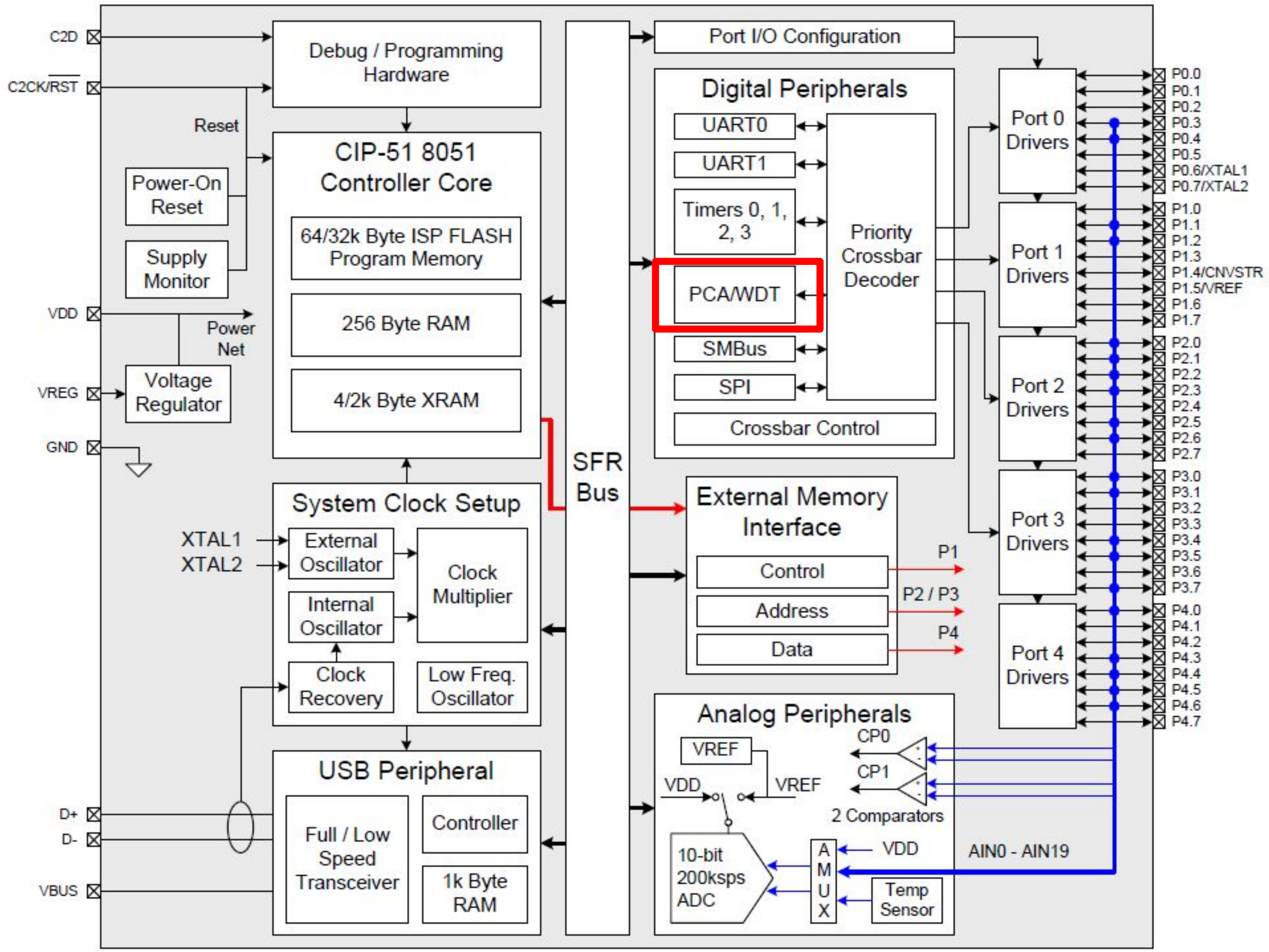
Observation table

Dout (O/P on LEDs)	**Vin (Calculated)**	**Vin (Measured/Given)**
0011101101	0.763V	
1110001100	2.92V	
0101100110	1.15V	
1100011100	2.56V	
1010011011	2.14V	
1111111111	3.29V	

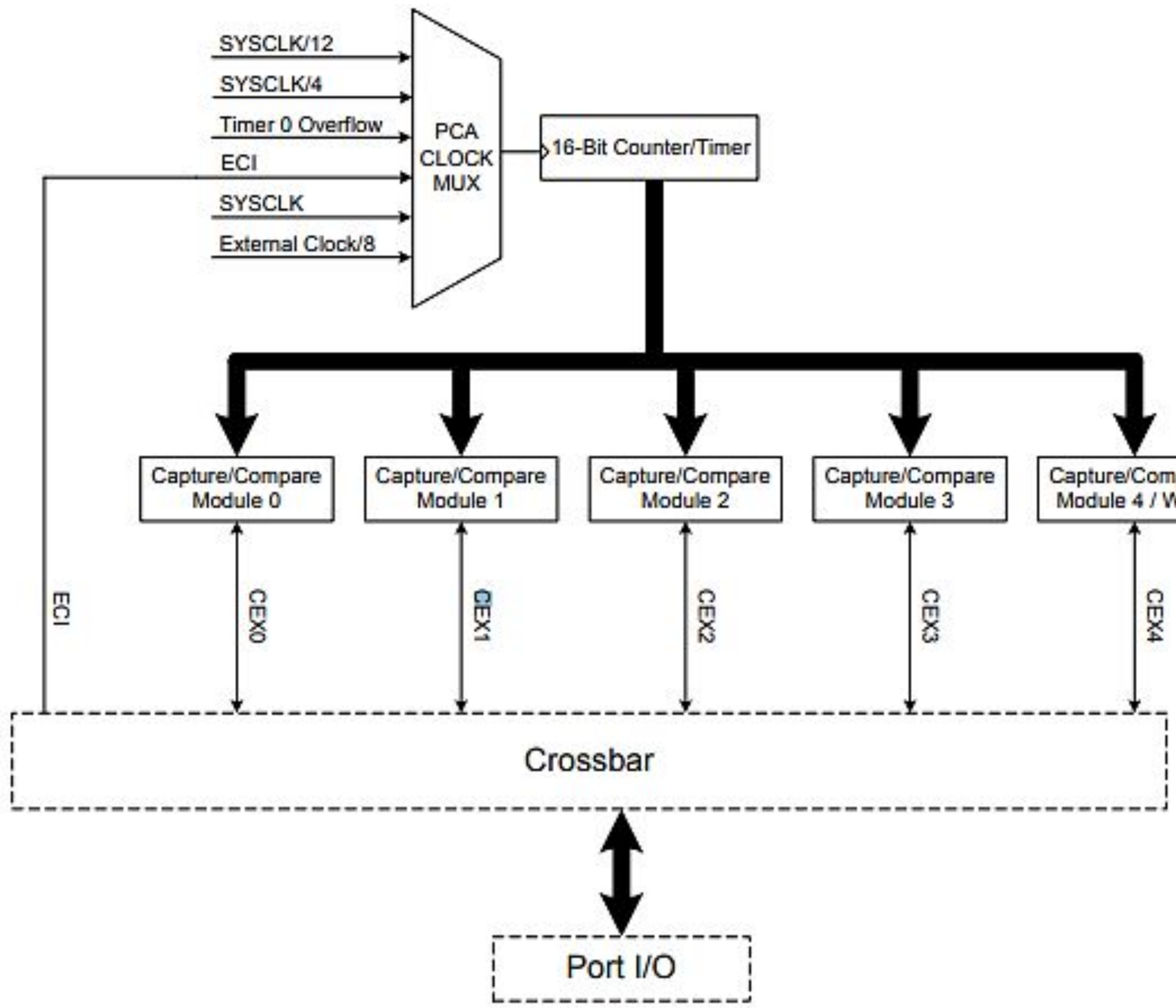
Programmable Counter Array (PCA0)

- The Programmable Counter Array (PCA0) provides enhanced timer functionality while requiring less CPU intervention than the standard 8051 counter/timers.
- The PCA consists of a dedicated 16-bit counter/timer and five 16-bit capture/compare modules.
- Each capture/compare module has its own associated I/O line (CEXn) which is routed through the Crossbar to Port I/O when enabled.

The Programmable Counter Array (PCA0)



PCA Block Diagram



Programmable Counter Array (PCA0)

- The counter/timer is driven by a programmable timebase that can select between six sources:
  - system clock
  - system clock divided by four
  - system clock divided by twelve
  - the external oscillator clock source divided by 8
  - Timer 0 overflow
  - an external clock signal on the ECI input pin

Programmable Counter Array (PCA0)

- Each capture/compare module may be configured to operate independently in one of six modes:
  - Edge-Triggered Capture
  - Software Timer
  - High-Speed Output
  - Frequency Output
  - 8-Bit PWM
  - 16-Bit PWM

Programmable Counter Array (PCA0)

- Each module can be used independently to generate a Pulse Width Modulated (PWM) output on its associated CEXn pin.

- The frequency of the output is dependent on the timebase for the PCA counter/timer.
  - **Calculations for PCA0L similar to timer calculation in Mode2.**

- The duty cycle of the PWM output signal is varied using the module's PCA0CPLn capture/compare register.

Timer 0 Mode 2 Block Diagram

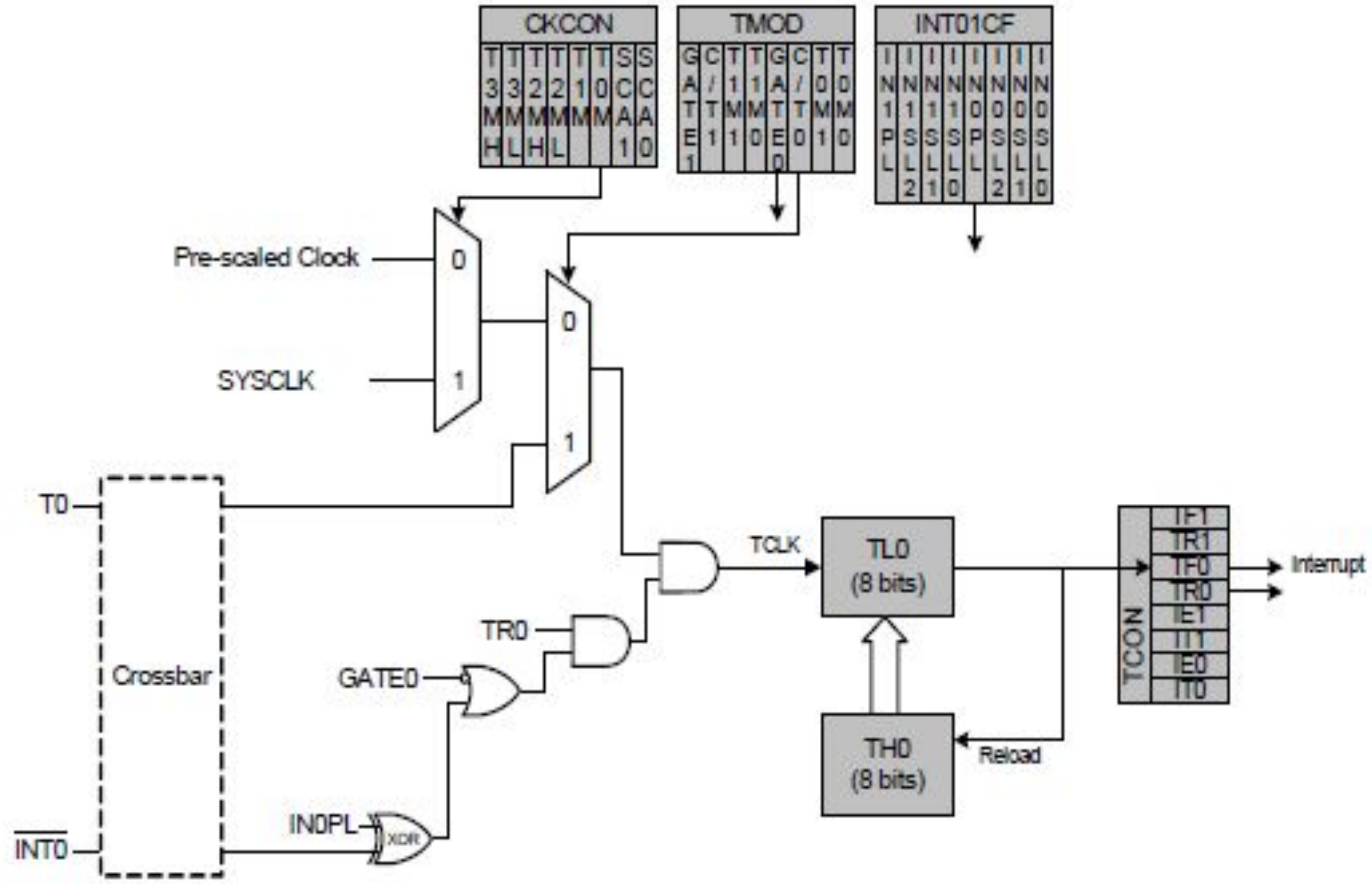
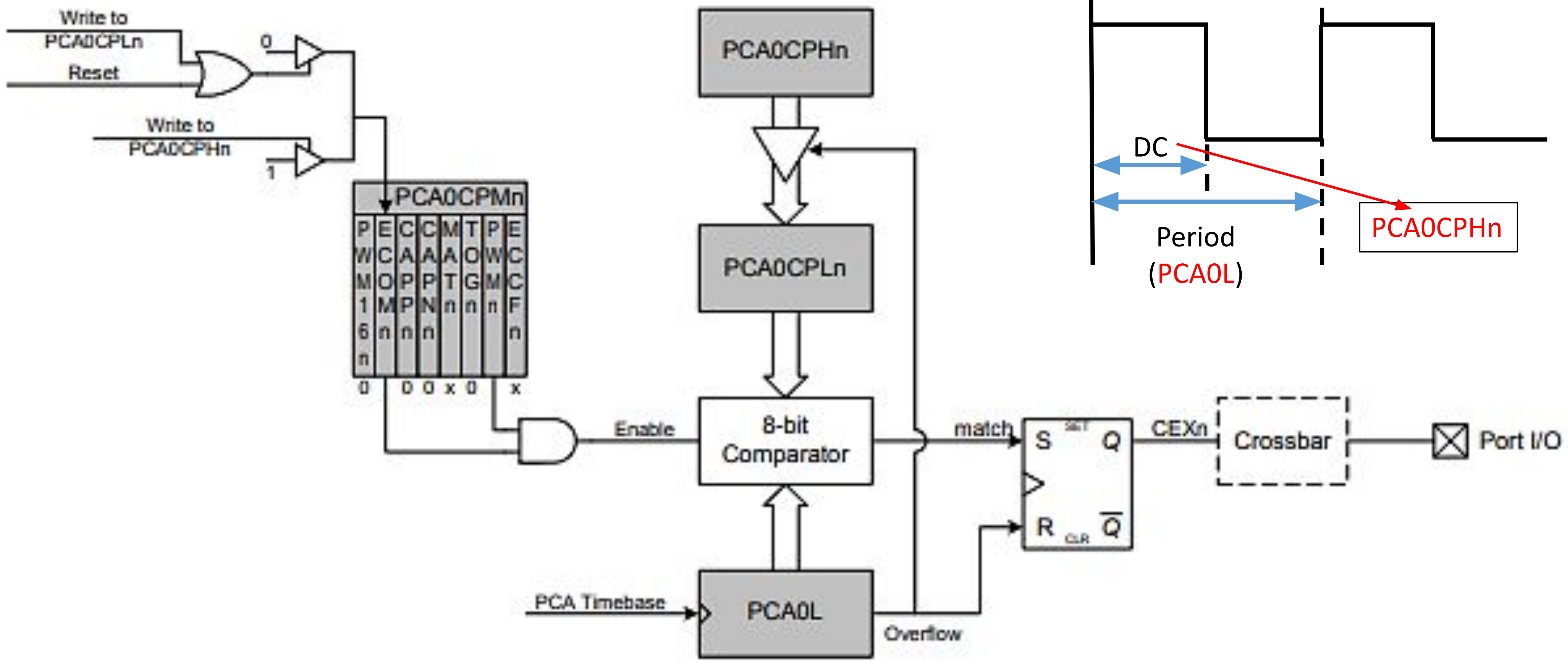
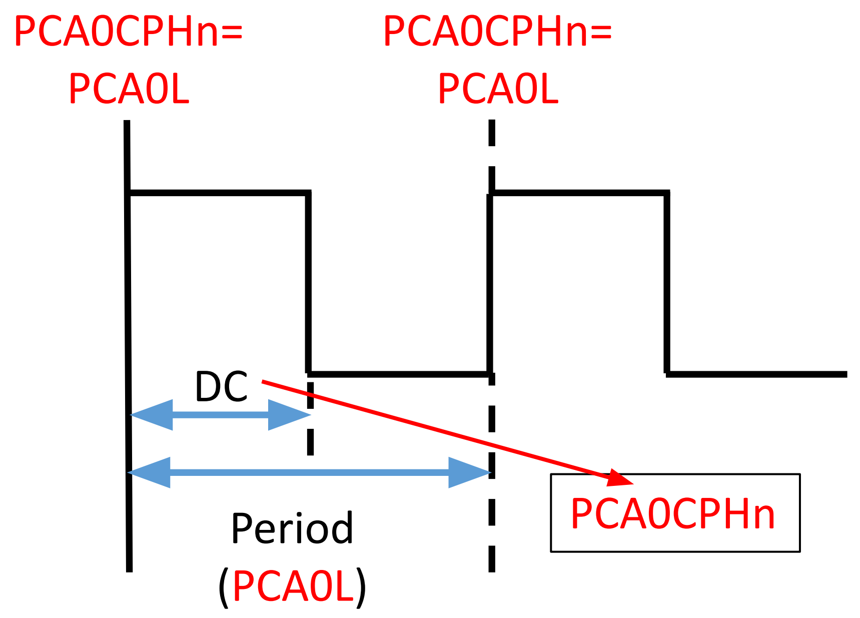


Figure 21.2. T0 Mode 2 Block Diagram

PCA 8-Bit PWM Mode Diagram





8-Bit Pulse Width Modulator Mode

- Setting the ECOMn and PWMn bits in the PCA0CPMn register enables 8-Bit Pulse Width Modulator mode.
- When the value in the low byte of the PCA counter/timer (PCA0L) is equal to the value in PCA0CPLn, the output on the CEXn pin will be set.
- When the count value in PCA0L overflows, the CEXn output will be reset.
- Also, when the counter/timer low byte (PCA0L) overflows from 0xFF to 0x00, PCA0CPLn is reloaded automatically with the value stored in the module's capture/compare high byte (PCA0CPHn) without software intervention.
- The duty cycle for 8-Bit PWM Mode is given by Equation

$$DutyCycle = \frac{(256 - PCA0CPHn)}{256}$$

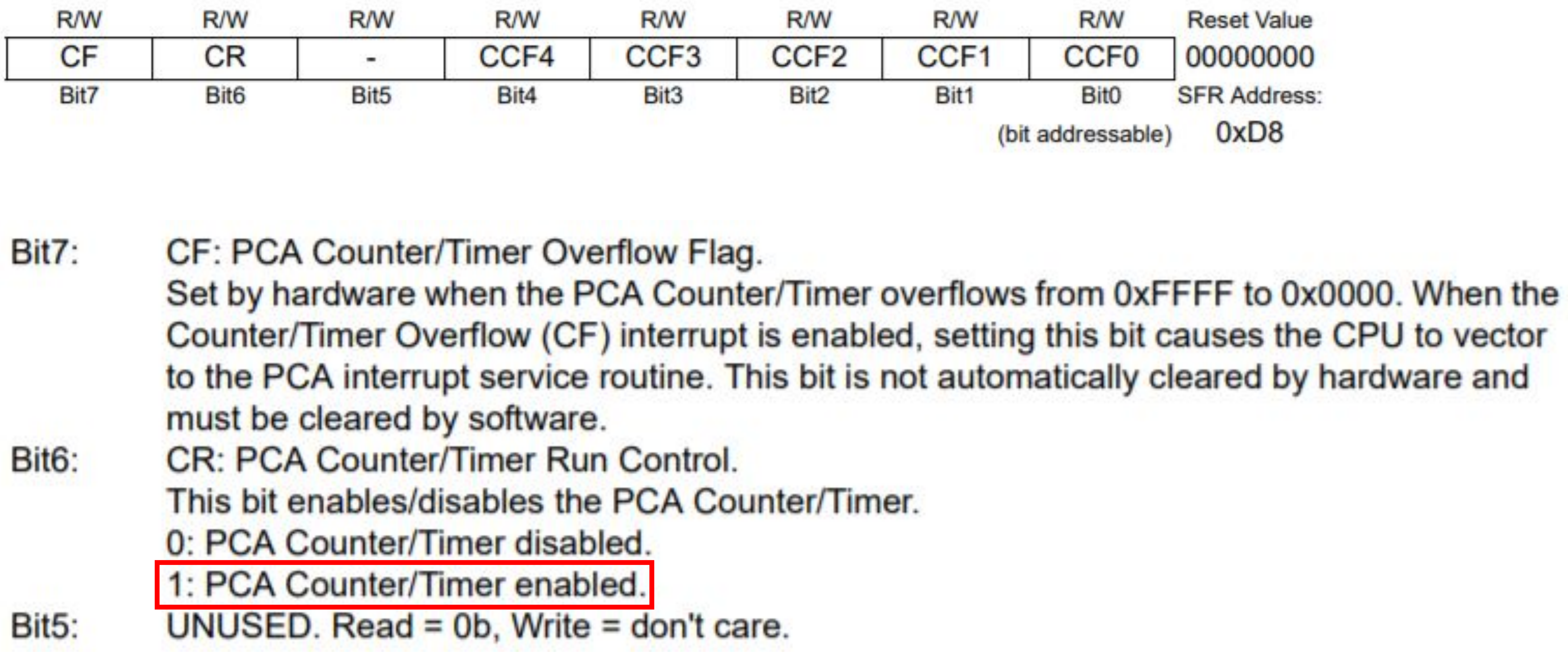
PCA0CPH0 = 256 – (256 * DutyCycle)

8-Bit Pulse Width Modulator Mode

- The largest duty cycle is 100% (PCA0CPHn = 0), and the smallest duty cycle is 0.39% (PCA0CPHn = 0xFF).

- A 0% duty cycle may be generated by clearing the ECOMn bit to ‘0’.

PCA0CN: PCA Control CR=1



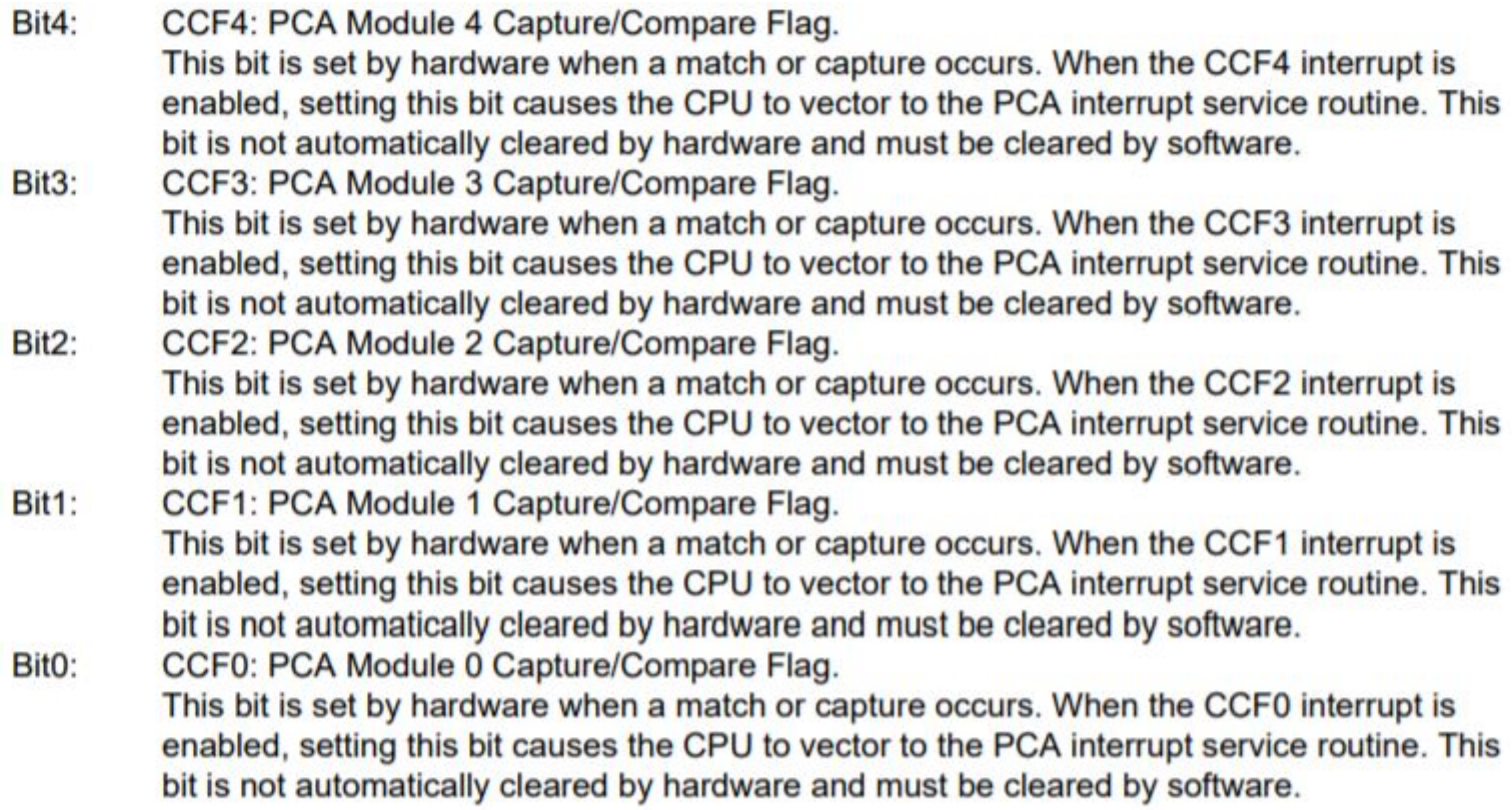
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
CF	CR	-	CCF4	CCF3	CCF2	CCF1	CCF0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:
							(bit addressable)	0xD8

Bit7: CF: PCA Counter/Timer Overflow Flag.
Set by hardware when the PCA Counter/Timer overflows from 0xFFFF to 0x0000. When the Counter/Timer Overflow (CF) interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.

Bit6: CR: PCA Counter/Timer Run Control.
This bit enables/disables the PCA Counter/Timer.
0: PCA Counter/Timer disabled.
$\boxed{\text{1: PCA Counter/Timer enabled.}}$

Bit5: UNUSED. Read = 0b, Write = don't care.

PCA0CN: PCA Control



Bit4: CCF4: PCA Module 4 Capture/Compare Flag.
This bit is set by hardware when a match or capture occurs. When the CCF4 interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.

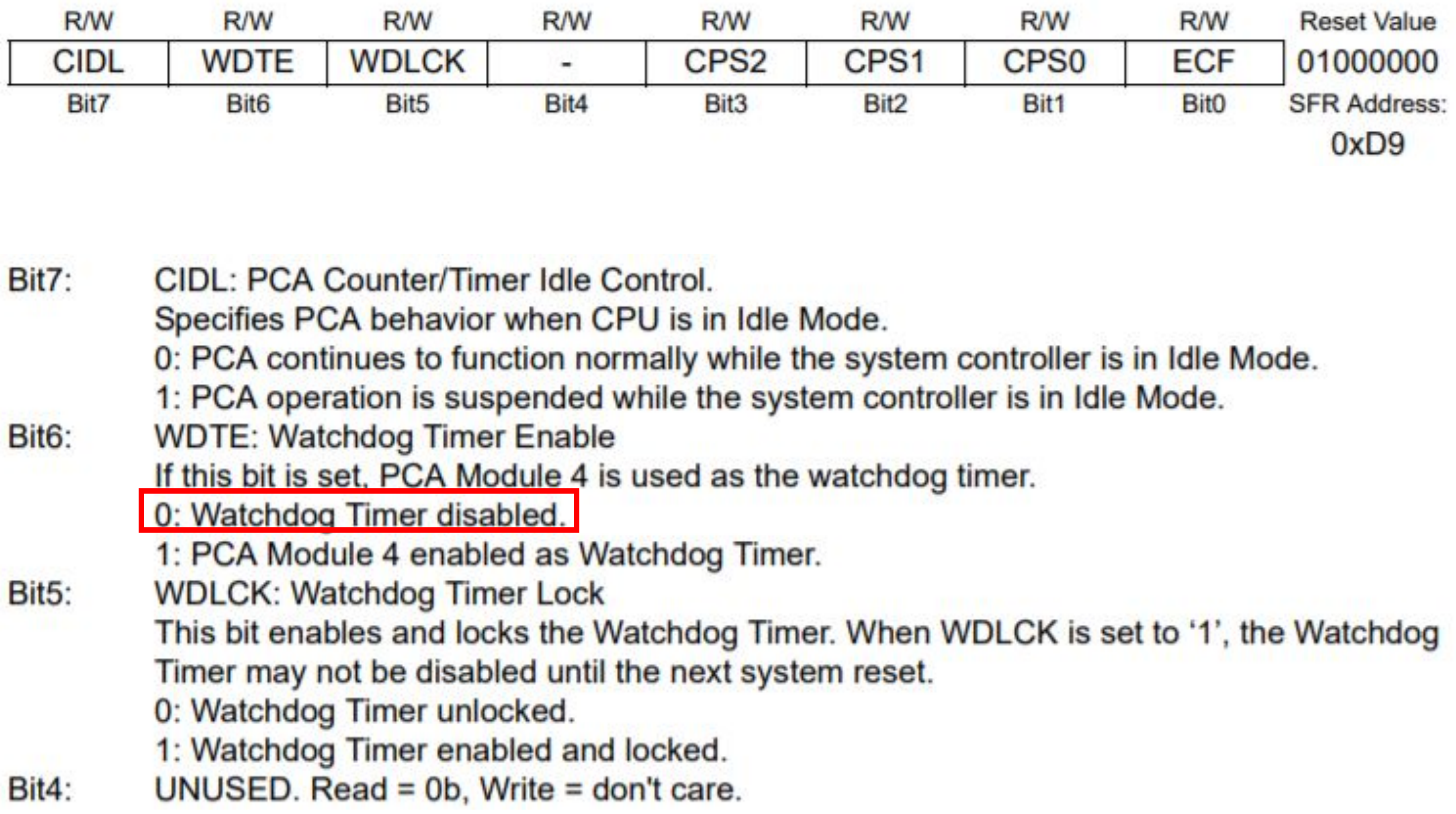
Bit3: CCF3: PCA Module 3 Capture/Compare Flag.
This bit is set by hardware when a match or capture occurs. When the CCF3 interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.

Bit2: CCF2: PCA Module 2 Capture/Compare Flag.
This bit is set by hardware when a match or capture occurs. When the CCF2 interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.

Bit1: CCF1: PCA Module 1 Capture/Compare Flag.
This bit is set by hardware when a match or capture occurs. When the CCF1 interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.

Bit0: CCF0: PCA Module 0 Capture/Compare Flag.
This bit is set by hardware when a match or capture occurs. When the CCF0 interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.

PCA0MD: PCA Mode (Selecting Timebase)



R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
CIDL	WDTE	WDLCK	-	CPS2	CPS1	CPS0	ECF	01000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xD9

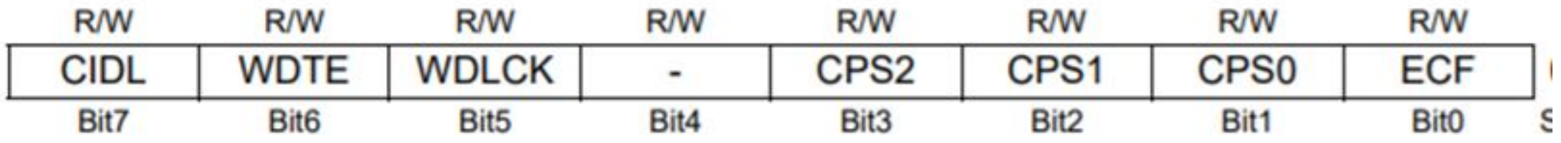
Bit7: CIDL: PCA Counter/Timer Idle Control.
Specifies PCA behavior when CPU is in Idle Mode.
0: PCA continues to function normally while the system controller is in Idle Mode.
1: PCA operation is suspended while the system controller is in Idle Mode.

Bit6: WDTE: Watchdog Timer Enable
If this bit is set, PCA Module 4 is used as the watchdog timer.
0: Watchdog Timer disabled.
1: PCA Module 4 enabled as Watchdog Timer.

Bit5: WDLCK: Watchdog Timer Lock
This bit enables and locks the Watchdog Timer. When WDLCK is set to '1', the Watchdog Timer may not be disabled until the next system reset.
0: Watchdog Timer unlocked.
1: Watchdog Timer enabled and locked.

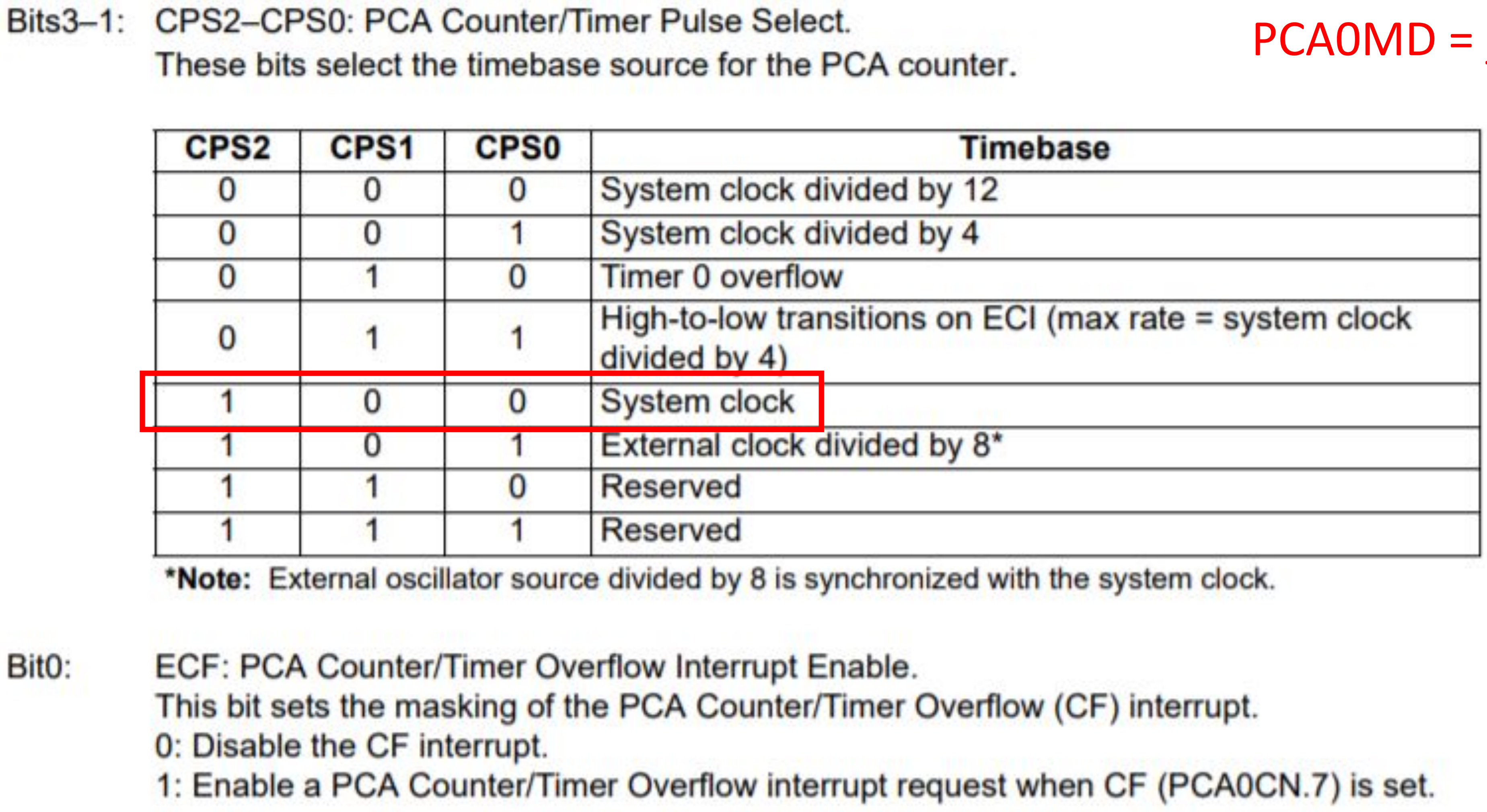
Bit4: UNUSED. Read = 0b, Write = don't care.

PCA0MD: PCA Mode



R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
CIDL	WDTE	WDLCK	-	CPS2	CPS1	CPS0	ECF
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

PCA0MD = _______



Bits3–1: CPS2–CPS0: PCA Counter/Timer Pulse Select.
These bits select the timebase source for the PCA counter.

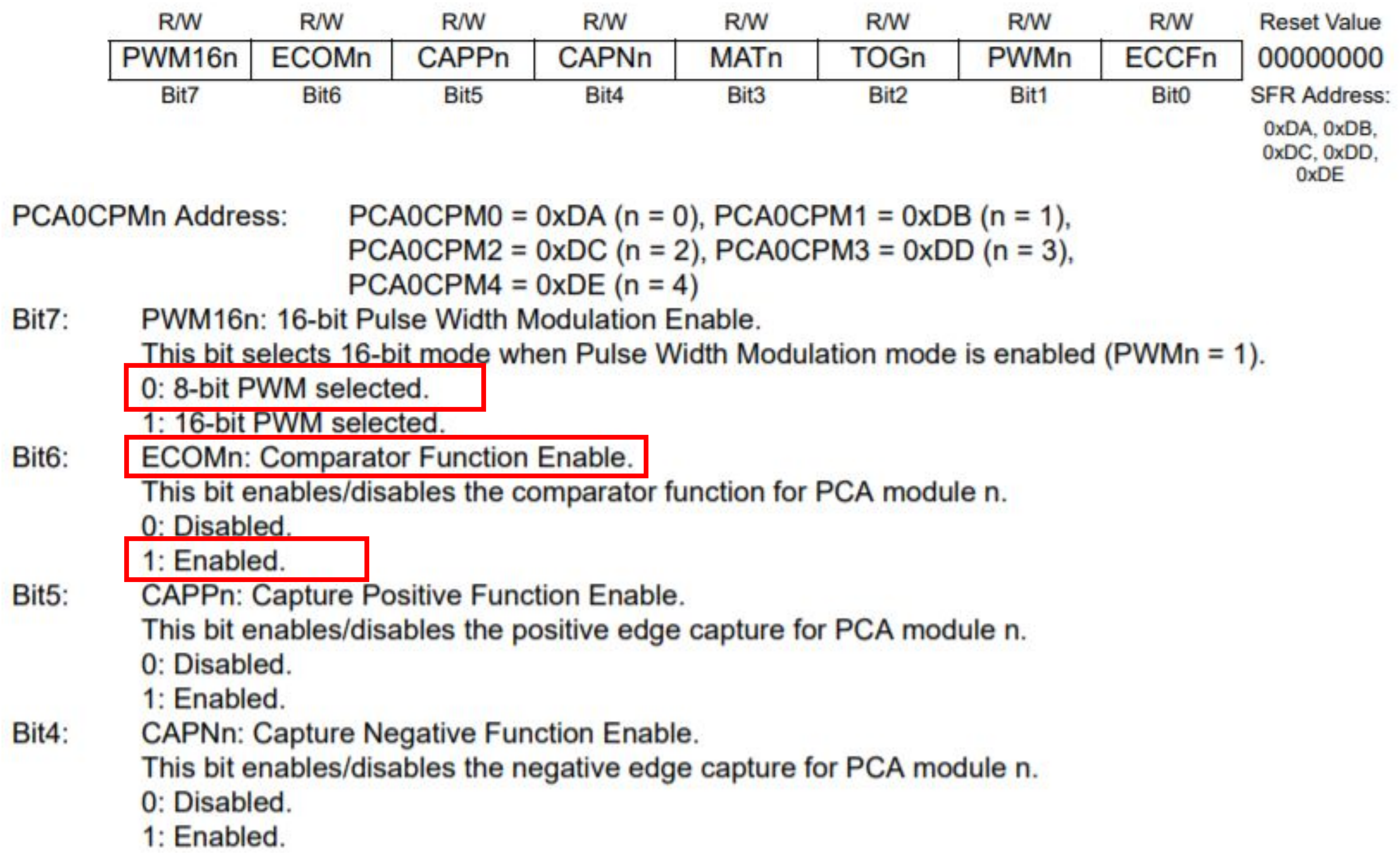
CPS2	CPS1	CPS0	Timebase
0	0	0	System clock divided by 12
0	0	1	System clock divided by 4
0	1	0	Timer 0 overflow
0	1	1	High-to-low transitions on ECI (max rate = system clock divided by 4)
1	0	0	System clock
1	0	1	External clock divided by 8*
1	1	0	Reserved
1	1	1	Reserved

***Note:** External oscillator source divided by 8 is synchronized with the system clock.

Bit0: ECF: PCA Counter/Timer Overflow Interrupt Enable.
This bit sets the masking of the PCA Counter/Timer Overflow (CF) interrupt.
0: Disable the CF interrupt.
1: Enable a PCA Counter/Timer Overflow interrupt request when CF (PCA0CN.7) is set.

Note: When the WDTE bit is set to ‘1’, the PCA0MD register cannot be modified. To change the contents of the PCA0MD register, the Watchdog Timer must first be disabled.

PCA0CPMn: PCA Capture/Compare Mode



R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
PWM16n	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xDA, 0xDB, 0xDC, 0xDD, 0xDE

PCA0CPMn Address: PCA0CPM0 = 0xDA (n = 0), PCA0CPM1 = 0xDB (n = 1), PCA0CPM2 = 0xDC (n = 2), PCA0CPM3 = 0xDD (n = 3), PCA0CPM4 = 0xDE (n = 4)

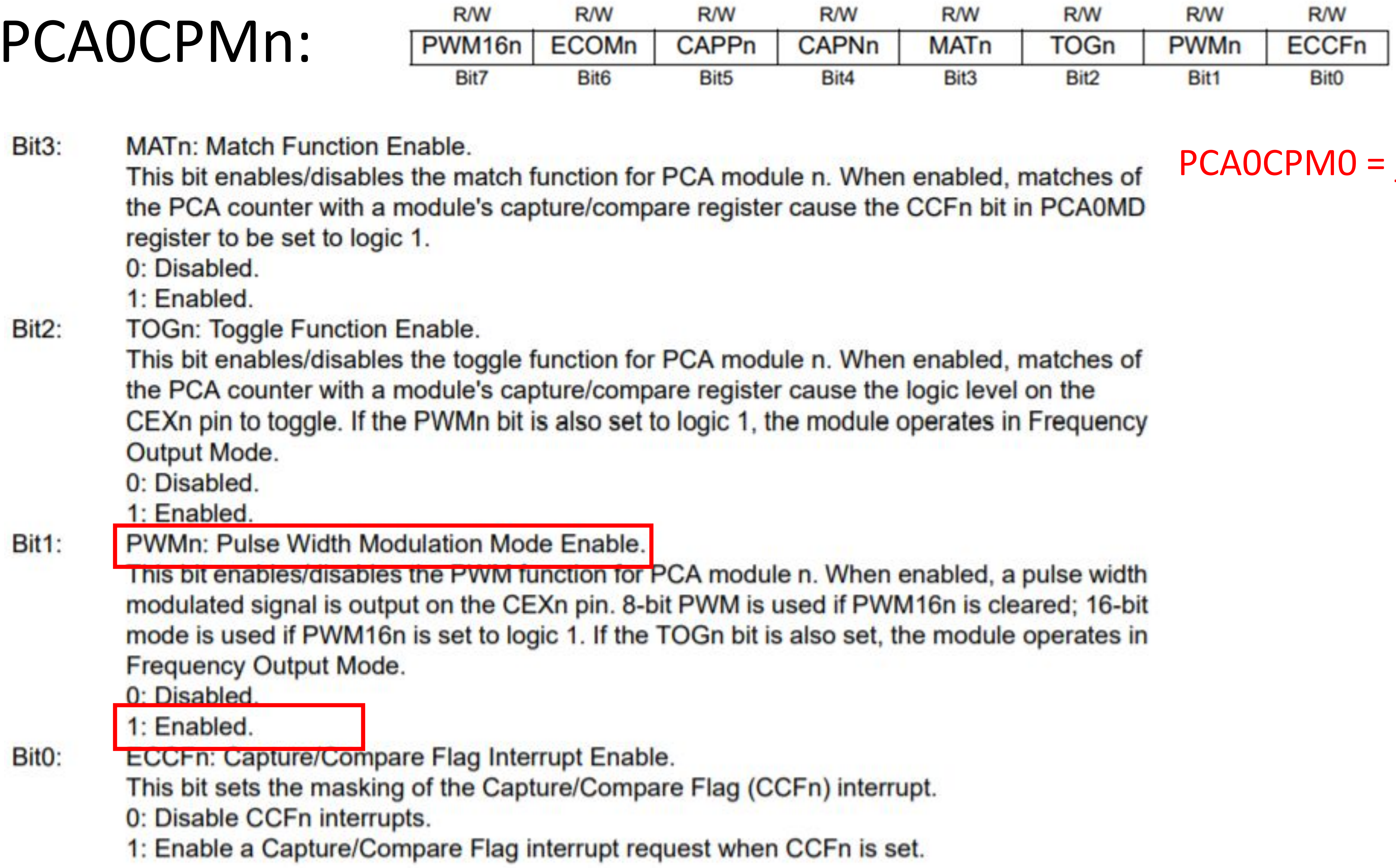
Bit7: PWM16n: 16-bit Pulse Width Modulation Enable.
This bit selects 16-bit mode when Pulse Width Modulation mode is enabled (PWMn = 1).
0: 8-bit PWM selected.
1: 16-bit PWM selected.

Bit6: ECOMn: Comparator Function Enable.
This bit enables/disables the comparator function for PCA module n.
0: Disabled.
1: Enabled.

Bit5: CAPPn: Capture Positive Function Enable.
This bit enables/disables the positive edge capture for PCA module n.
0: Disabled.
1: Enabled.

Bit4: CAPNn: Capture Negative Function Enable.
This bit enables/disables the negative edge capture for PCA module n.
0: Disabled.
1: Enabled.

PCA0CPMn:



R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PWM16n	ECOMn	CAPPn	CAPNn	MATn	TOGn	PWMn	ECCFn
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

Bit3: MATn: Match Function Enable.
This bit enables/disables the match function for PCA module n. When enabled, matches of the PCA counter with a module's capture/compare register cause the CCFn bit in PCA0MD register to be set to logic 1.
0: Disabled.
1: Enabled.

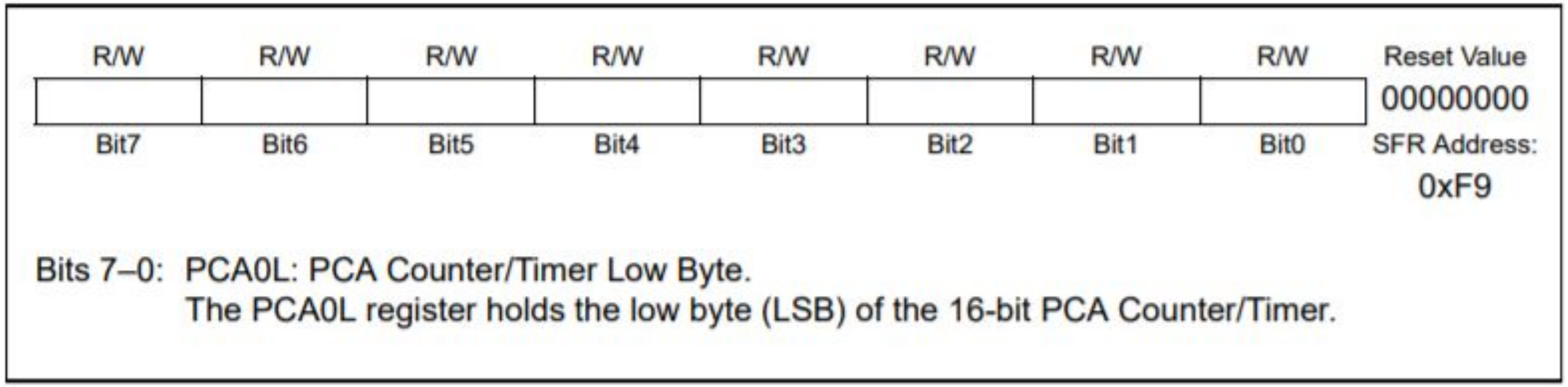
Bit2: TOGn: Toggle Function Enable.
This bit enables/disables the toggle function for PCA module n. When enabled, matches of the PCA counter with a module's capture/compare register cause the logic level on the CEXn pin to toggle. If the PWMn bit is also set to logic 1, the module operates in Frequency Output Mode.
0: Disabled.
1: Enabled.

Bit1: PWMn: Pulse Width Modulation Mode Enable.
This bit enables/disables the PWM function for PCA module n. When enabled, a pulse width modulated signal is output on the CEXn pin. 8-bit PWM is used if PWM16n is cleared; 16-bit mode is used if PWM16n is set to logic 1. If the TOGn bit is also set, the module operates in Frequency Output Mode.
0: Disabled.
1: Enabled.

Bit0: ECCFn: Capture/Compare Flag Interrupt Enable.
This bit sets the masking of the Capture/Compare Flag (CCFn) interrupt.
0: Disable CCFn interrupts.
1: Enable a Capture/Compare Flag interrupt request when CCFn is set.

PCA0CPM0 = ______

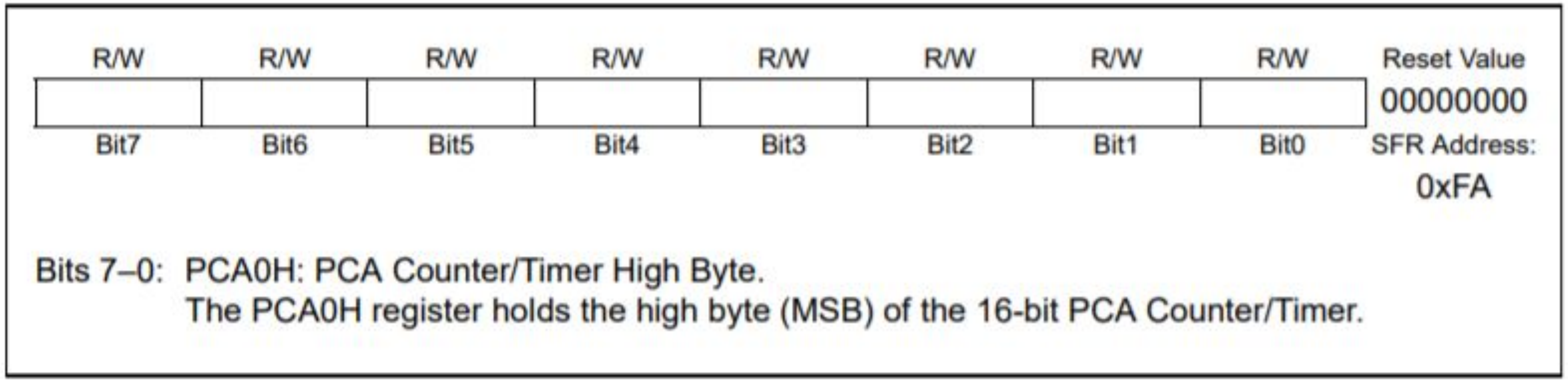
PCA0L: PCA Counter/Timer Low Byte



R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xF9

Bits 7–0: PCA0L: PCA Counter/Timer Low Byte.
The PCA0L register holds the low byte (LSB) of the 16-bit PCA Counter/Timer.

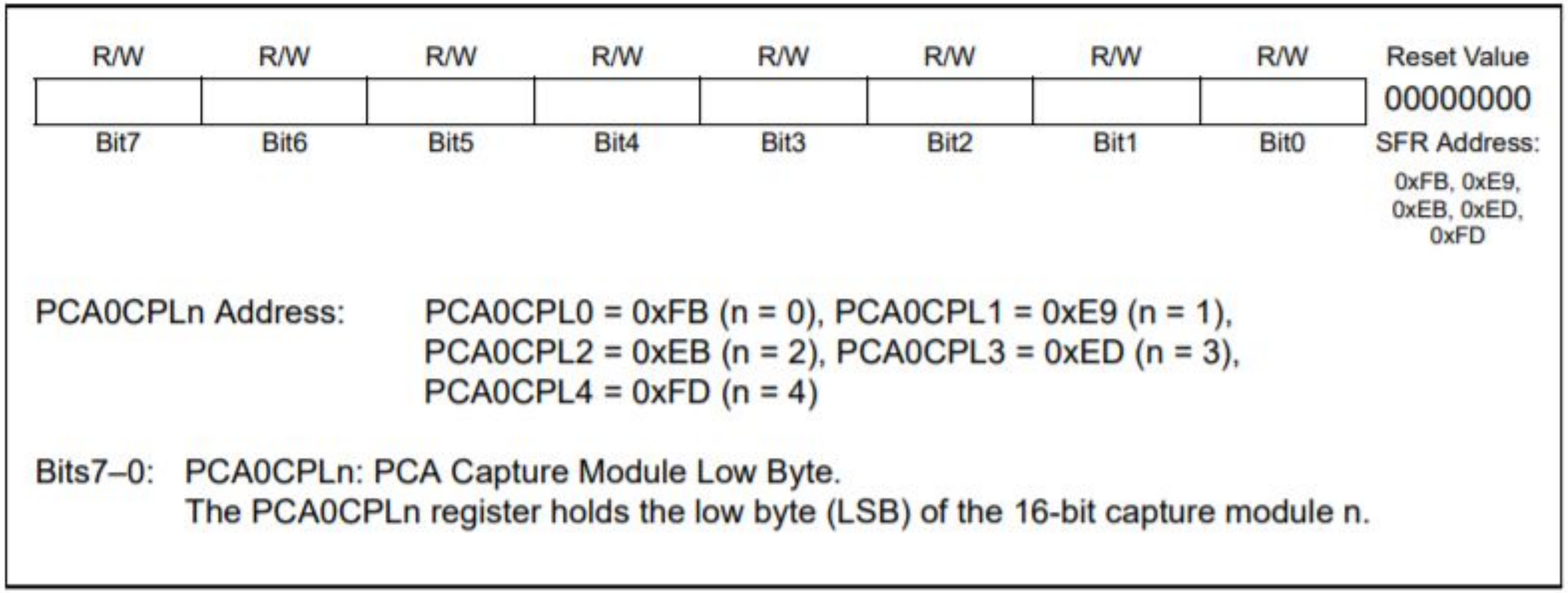
PCA0H: PCA Counter/Timer High Byte



R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xFA

Bits 7–0: PCA0H: PCA Counter/Timer High Byte.
The PCA0H register holds the high byte (MSB) of the 16-bit PCA Counter/Timer.

PCA0CPLn: PCA Capture Module Low Byte

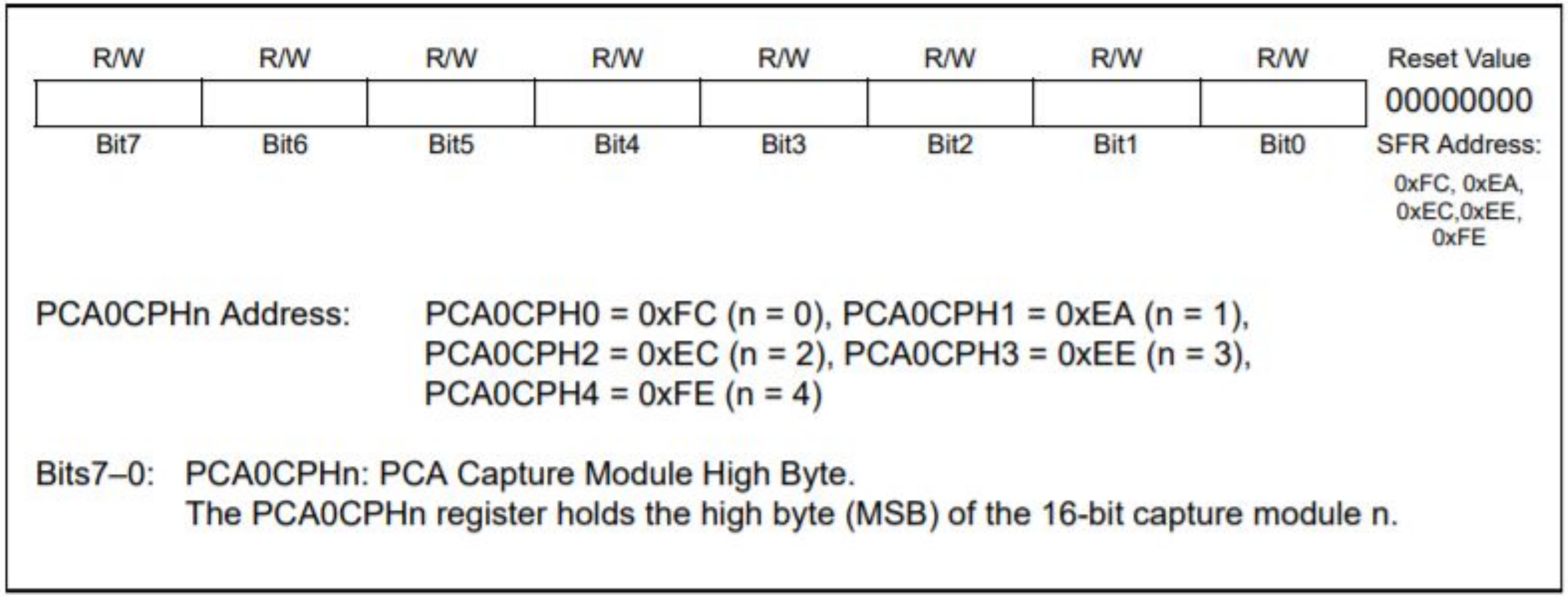


R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xFB, 0xE9, 0xEB, 0xED, 0xFD

PCA0CPLn Address: PCA0CPL0 = 0xFB (n = 0), PCA0CPL1 = 0xE9 (n = 1),
PCA0CPL2 = 0xEB (n = 2), PCA0CPL3 = 0xED (n = 3),
PCA0CPL4 = 0xFD (n = 4)

Bits7–0: PCA0CPLn: PCA Capture Module Low Byte.
The PCA0CPLn register holds the low byte (LSB) of the 16-bit capture module n.

PCA0CPHn: PCA Capture Module High Byte

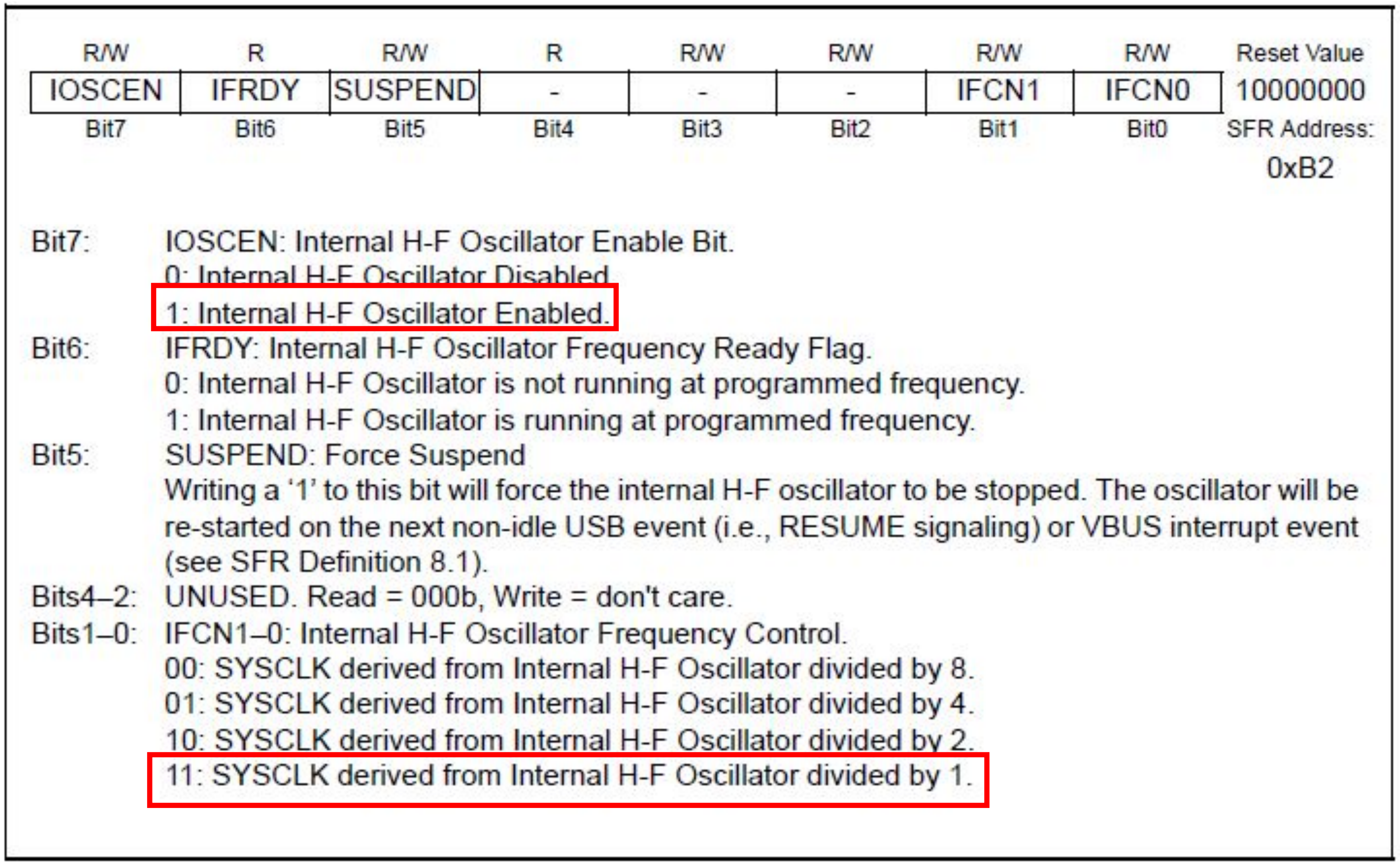


R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:
								0xFC, 0xEA, 0xEC,0xEE, 0xFE

PCA0CPHn Address: PCA0CPH0 = 0xFC (n = 0), PCA0CPH1 = 0xEA (n = 1),
PCA0CPH2 = 0xEC (n = 2), PCA0CPH3 = 0xEE (n = 3),
PCA0CPH4 = 0xFE (n = 4)

Bits7–0: PCA0CPHn: PCA Capture Module High Byte.
The PCA0CPHn register holds the high byte (MSB) of the 16-bit capture module n.

OSCICN - Internal H-F Oscillator Control - internal oscillator to run at its maximum frequency

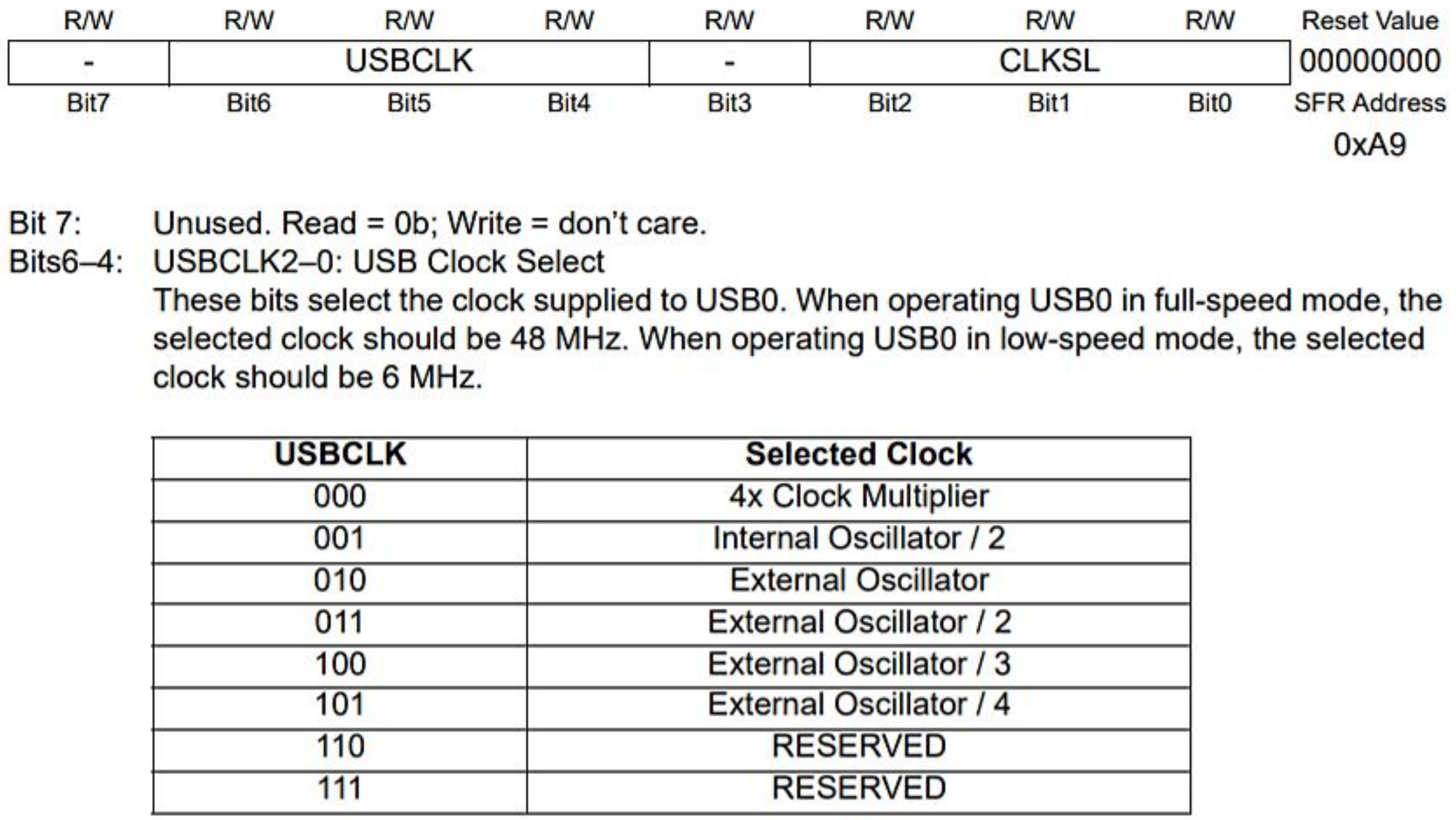


R/W	R	R/W	R	R/W	R/W	R/W	R/W	Reset Value
IOSCEN	IFRDY	SUSPEND	-	-	-	IFCN1	IFCN0	10000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xB2

Bit7:	IOSCEN: Internal H-F Oscillator Enable Bit. 0: Internal H-F Oscillator Disabled. 1: Internal H-F Oscillator Enabled.
Bit6:	IFRDY: Internal H-F Oscillator Frequency Ready Flag. 0: Internal H-F Oscillator is not running at programmed frequency. 1: Internal H-F Oscillator is running at programmed frequency.
Bit5:	SUSPEND: Force Suspend Writing a '1' to this bit will force the internal H-F oscillator to be stopped. The oscillator will be re-started on the next non-idle USB event (i.e., RESUME signaling) or VBUS interrupt event (see SFR Definition 8.1).
Bits4–2:	UNUSED. Read = 000b, Write = don't care.
Bits1–0:	IFCN1–0: Internal H-F Oscillator Frequency Control. 00: SYSCLK derived from Internal H-F Oscillator divided by 8. 01: SYSCLK derived from Internal H-F Oscillator divided by 4. 10: SYSCLK derived from Internal H-F Oscillator divided by 2. 11: SYSCLK derived from Internal H-F Oscillator divided by 1.

OSCICN = ______

SFR: CLKSEL - Clock Select – To select system clock



R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
-	USBCLK			-	CLKSL			00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address
								0xA9

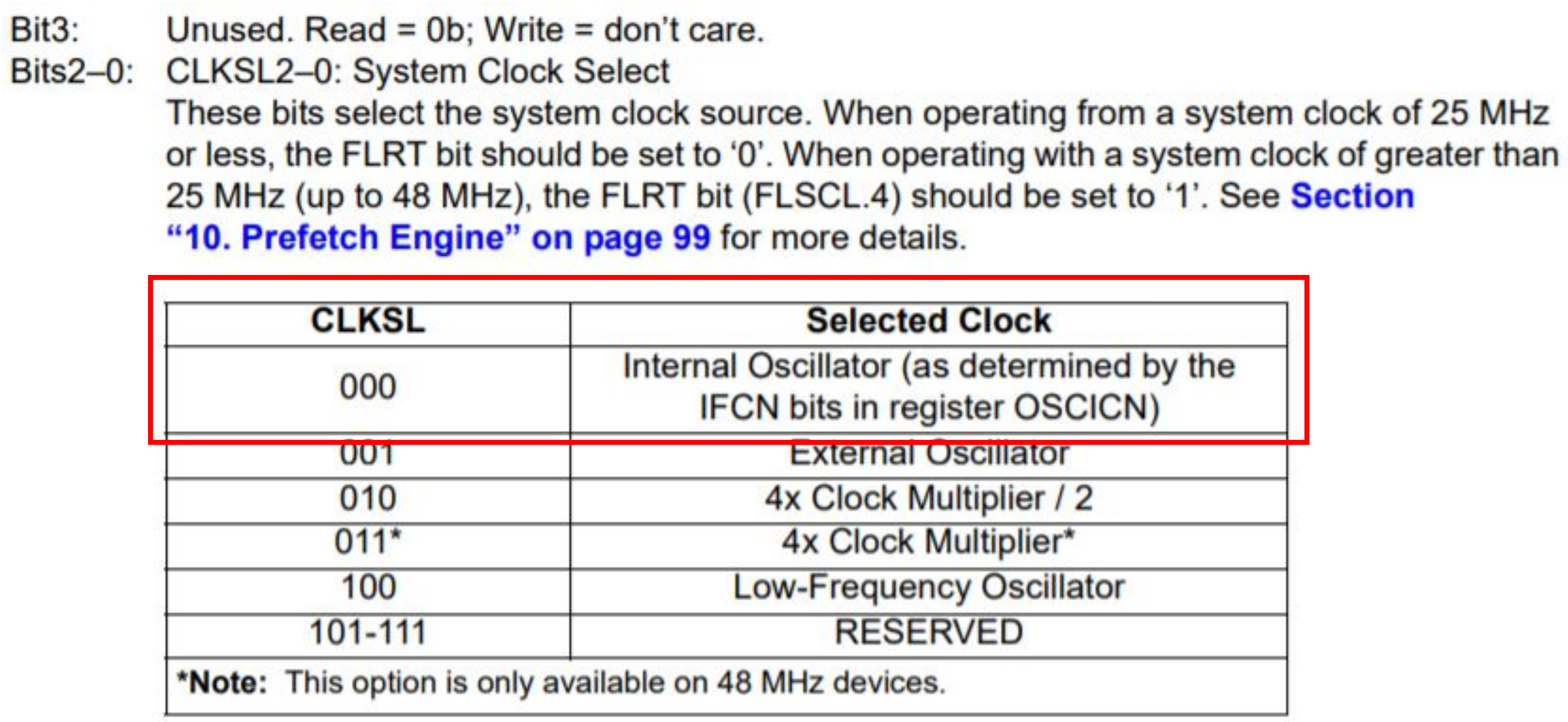
Bit 7: Unused. Read = 0b; Write = don't care.

Bits6–4: USBCLK2–0: USB Clock Select

These bits select the clock supplied to USB0. When operating USB0 in full-speed mode, the selected clock should be 48 MHz. When operating USB0 in low-speed mode, the selected clock should be 6 MHz.

USBCLK	Selected Clock
000	4x Clock Multiplier
001	Internal Oscillator / 2
010	External Oscillator
011	External Oscillator / 2
100	External Oscillator / 3
101	External Oscillator / 4
110	RESERVED
111	RESERVED

SFR: CLKSEL - Clock Select



Bit3: Unused. Read = 0b; Write = don't care.

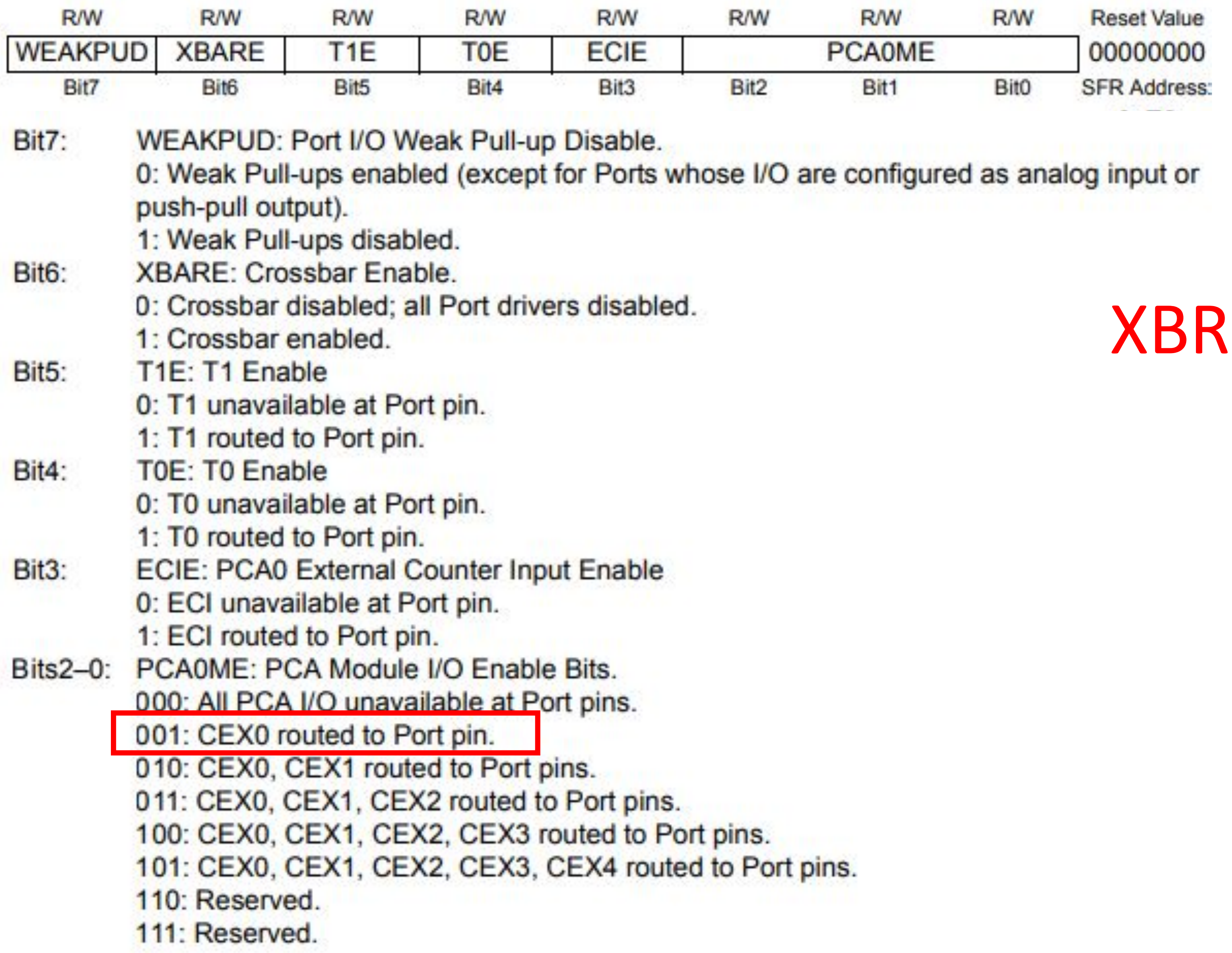
Bits2–0: CLKSL2–0: System Clock Select

These bits select the system clock source. When operating from a system clock of 25 MHz or less, the FLRT bit should be set to '0'. When operating with a system clock of greater than 25 MHz (up to 48 MHz), the FLRT bit (FLSCL.4) should be set to '1'. See **Section "10. Prefetch Engine" on page 99** for more details.

CLKSL	Selected Clock
000	Internal Oscillator (as determined by the IFCN bits in register OSCICN)
001	External Oscillator
010	4x Clock Multiplier / 2
011*	4x Clock Multiplier*
100	Low-Frequency Oscillator
101-111	RESERVED
***Note:** This option is only available on 48 MHz devices.	

CLKSEL = __________

XBR1 (Crossbar Register 1) – To enable crossbar and route CEX0 t0 port pin



R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
WEAKPUD	XBARE	T1E	T0E	ECIE	PCA0ME			00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:

Bit7: WEAKPUD: Port I/O Weak Pull-up Disable.
0: Weak Pull-ups enabled (except for Ports whose I/O are configured as analog input or push-pull output).
1: Weak Pull-ups disabled.

Bit6: XBARE: Crossbar Enable.
0: Crossbar disabled; all Port drivers disabled.
1: Crossbar enabled.

Bit5: T1E: T1 Enable
0: T1 unavailable at Port pin.
1: T1 routed to Port pin.

Bit4: T0E: T0 Enable
0: T0 unavailable at Port pin.
1: T0 routed to Port pin.

Bit3: ECIE: PCA0 External Counter Input Enable
0: ECI unavailable at Port pin.
1: ECI routed to Port pin.

Bits2–0: PCA0ME: PCA Module I/O Enable Bits.
000: All PCA I/O unavailable at Port pins.
$\boxed{\text{001: CEX0 routed to Port pin.}}$
010: CEX0, CEX1 routed to Port pins.
011: CEX0, CEX1, CEX2 routed to Port pins.
100: CEX0, CEX1, CEX2, CEX3 routed to Port pins.
101: CEX0, CEX1, CEX2, CEX3, CEX4 routed to Port pins.
110: Reserved.
111: Reserved.

XBR1 = ______

PWM Generation Algorithm

1) Set internal oscillator to run at its maximum frequency (OSCICN)
2) Select the internal osc. as the SYSCLK source (CLKSEL)
3) Route CEX0 to P0.0, Enable crossbar and weak pull-up (XBR1)
4) Set CEX0 (P1.4) to push-pull (P1MDOUT)
5) Disable watchdog timer and Use SYSCLK as time base (PCA0MD)
6) Enable Comparator Function and Pulse Width Modulation Mode (PCA0CPM0)
7) Configure PWM duty cycle in PCA0CPH0
8) Start PCA counter (CR bit in PCA0CN)
9) Repeat steps 5 to 9

PWM Generation Program-P1.4

```
#include <c8051f340.h>
#define SYSCLK      12000000          // Internal oscillator frequency in Hz
void main (void)
{

   OSCICN = 0x83;                     // Set internal oscillator to run at its maximum frequency
   CLKSEL = 0x00;                     // Select the internal osc. as the SYSCLK source
   XBR1    = 0x41;                    // Route CEX0 to P0.0, Enable crossbar and weak pull-up
   P1MDOUT = 0x10;                    // Set CEX0 (P1.4) to push-pull
   P0SKIP = 0xff;
   P1SKIP = 0x0f;
   while (1)
   {
      PCA0MD = 0x08;                  // Use SYSCLK as time base
      PCA0CPM0 = 0x42;                // Module 0 = 8-bit PWM mode
      PCA0CPH0 = 256 - (256 * 0.50);    // Configure initial PWM duty cycle = 50%
      CR = 1;                         // Start PCA counter
   }
}
```

PWM Calculations

Procedure for **Time to Count** Calculation– For PWM

Time period of 1clock cycle = (1/SystemClock)

Count = (Require time period/Time period of 1 clock cycle)

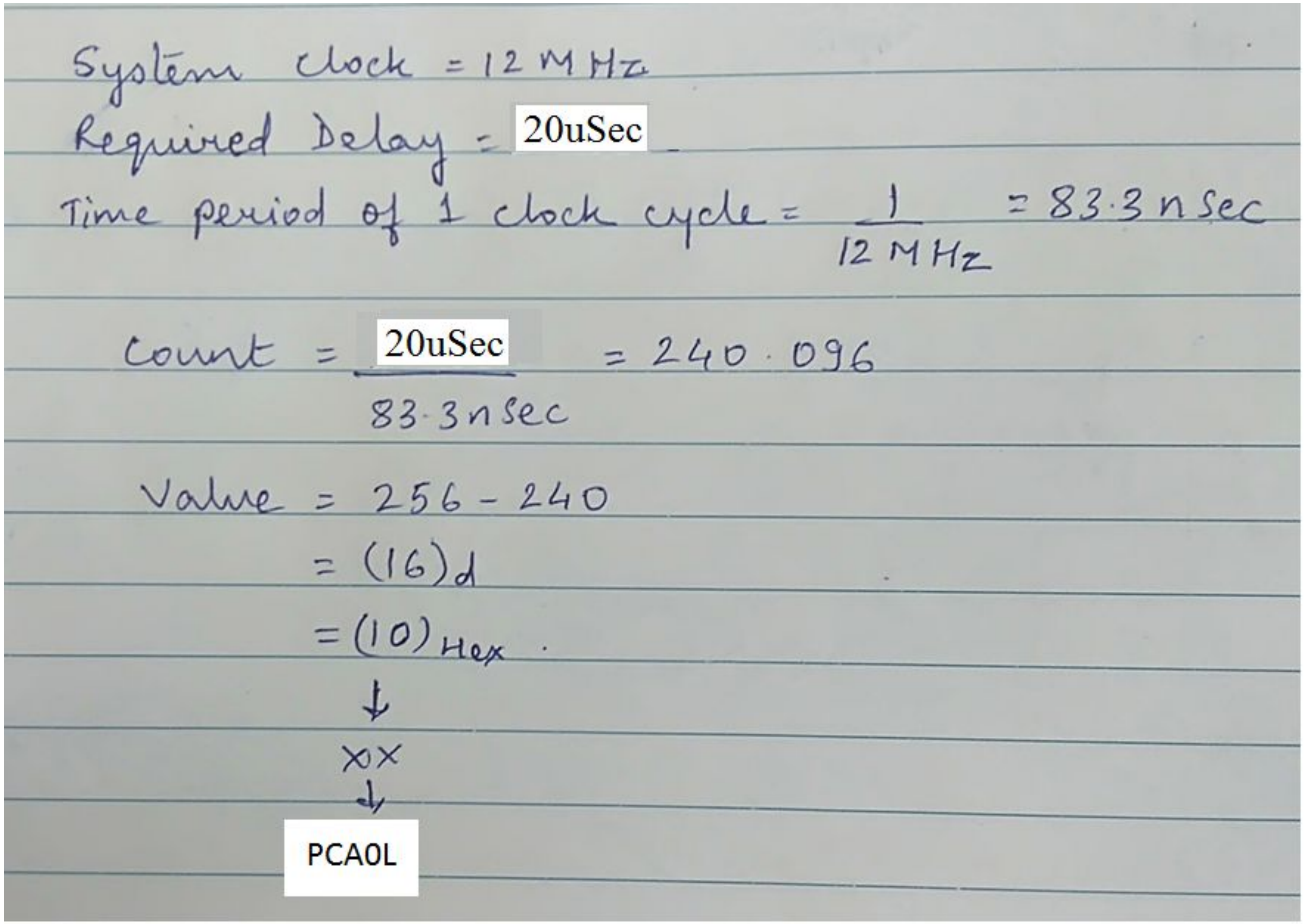
Value to be loaded in timer register,
Value (in decimal)= 256-Count

Convert Value from decimal to hex (XX)

Load XX in PCA0L register

Example for Time to Count Calculation for PWM

Desired Period= 20uSec



System clock = 12 MHz

Required Delay = 20uSec

Time period of 1 clock cycle = $\frac{1}{12\text{ MHz}}$ = 83.3 n Sec

Count = $\frac{20\text{uSec}}{83.3\text{ n Sec}}$ = 240.096

Value = 256 − 240

= $(16)_d$

= $(10)_{Hex}$.

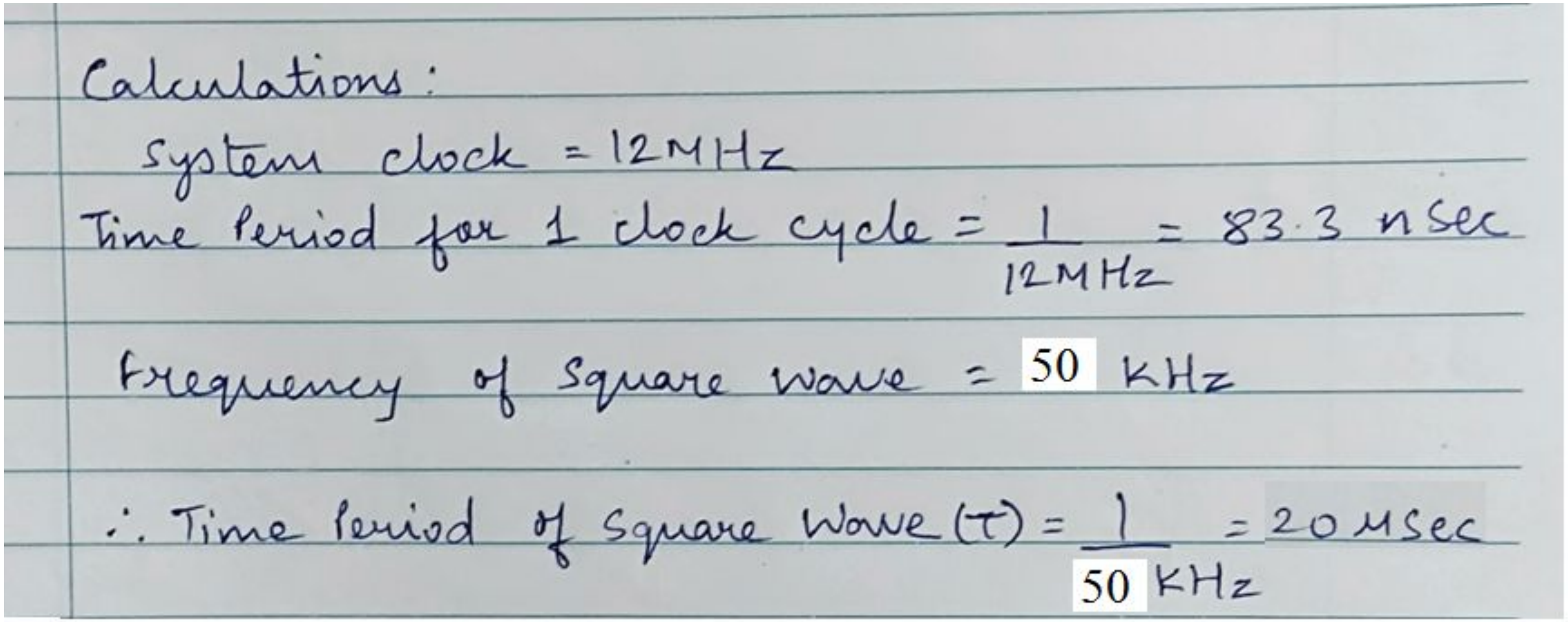
$\downarrow$

XX

$\downarrow$

PCA0L

Write an Embedded C program for generation of 50KHz PWM waveform with 50% duty cycle on Pin P2.3. Assume system clock = 12MHz.



Calculations :

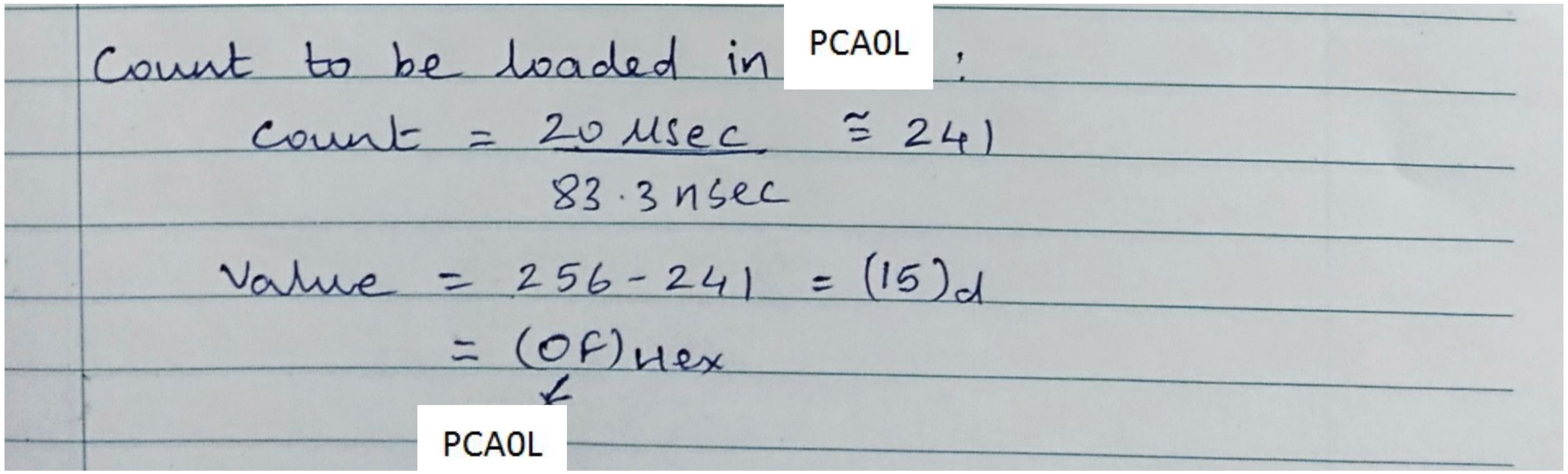
System clock = 12MHz

Time Period for 1 clock cycle $= \frac{1}{12MHz} = 83.3$ n Sec

Frequency of Square wave = 50 KHz

$\therefore$ Time Period of Square Wave $(T) = \frac{1}{50 KHz} = 20\ \mu$Sec

Contd.



Count to be loaded in PCA0L :

$$\text{count} = \frac{20\ \mu\text{sec}}{83.3\ \text{nsec}} \cong 241$$

$$\text{Value} = 256 - 241 = (15)_d$$

$$= (0F)_{Hex}$$

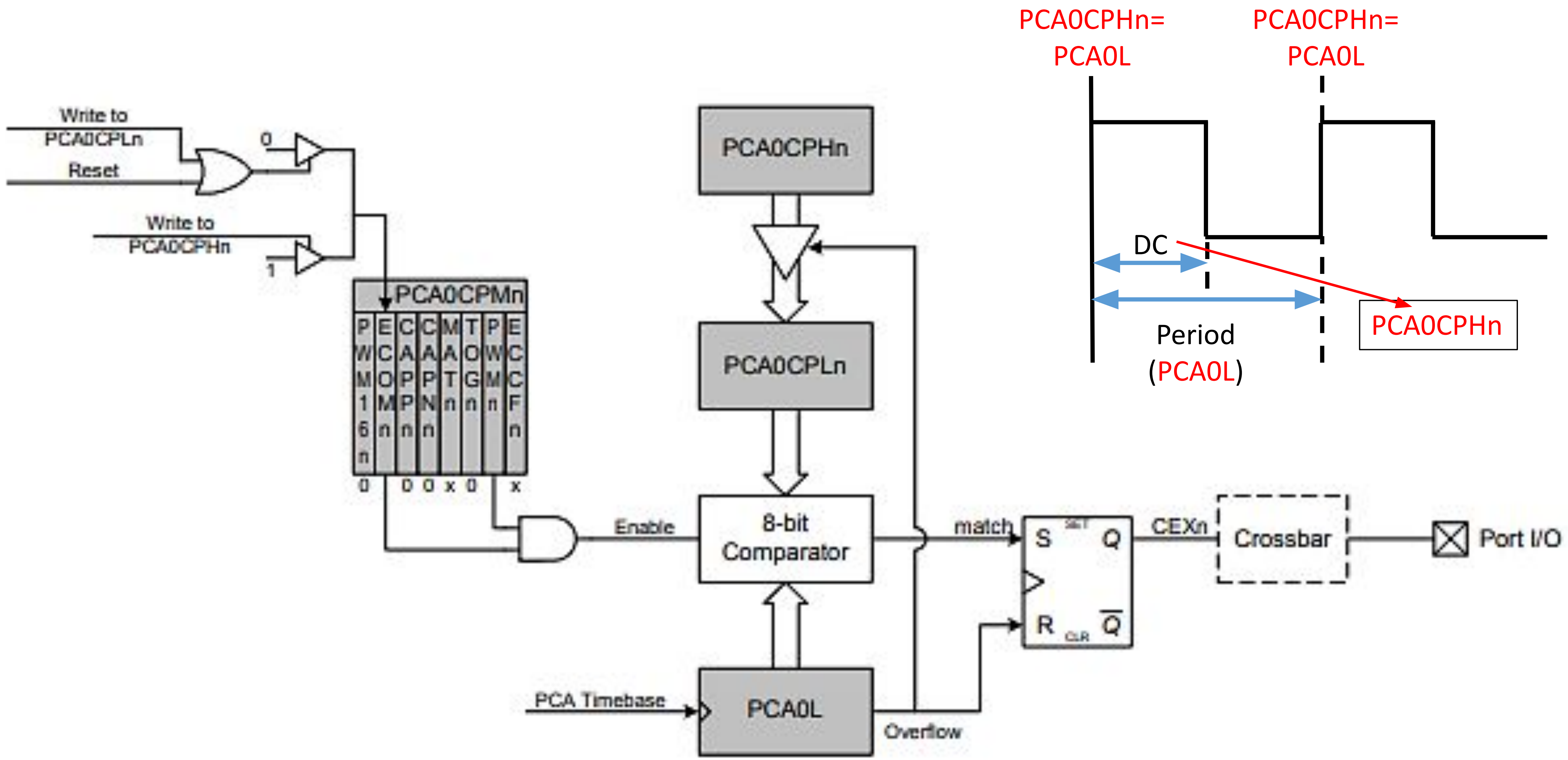
$\swarrow$

PCA0L

PWM Generation Program-P2.3

```
#include <c8051f340.h>
#define SYSCLK      12000000        // Internal oscillator frequency in Hz
void main (void)
{
  PCA0L = 0x0F;                     // Load count corresponding to desired frequency
  OSCICN = 0x83;                       // Set internal oscillator to run at its maximum frequency
  CLKSEL = 0x00;                      // Select the internal osc. as the SYSCLK source
  XBR1    = 0x41;                     // Route CEX0 to P0.0, Enable crossbar and weak pull-up
  P2MDOUT = 0x40;                    // Set CEX0 (P2.3) to push-pull
  P0SKIP = 0xFF;           // Skip Port 0
  P1SKIP = 0xFF;          // Skip Port 1

  P2SKIP = 0x07;          // Skip lower three pins of Port 2 as output required on P2.3
  while (1)
  {
     PCA0MD = 0x08;                        // Use SYSCLK as time base
     PCA0CPM0 = 0x42;                     // Module 0 = 8-bit PWM mode
     PCA0CPH0 = 256 - (256 * 0.50);     // Configure initial PWM duty cycle = 50%
     CR = 1;                              // Start PCA counter
  }
}
```

PCA 8-Bit PWM Mode Diagram



Find the value to be loaded in PCA0L for generating the PWM waveform of following frequencies:

Desired Frequency	PCA0L
60 KHz	38H
100KHz	88H
140KHz	AAH

PCA Observation table and calculation

- System clock=12MHz; Time for 1 clock cycle=83.3nsec
- Time Period=(256 – PCA0L)* Time for 1 clock cycle

Time Base	Duty Cycle (DC)	Time Period Measured on DSO	Time Period Calculated (HomeWork)	PCA0CPH0= 256-(256*DC)
System Clk PCA0MD=0x08	25% 50% 75%			
System Clk/12 PCA0MD=0x00	25% 50% 75%			
System Clk/4 PCA0MD=0x02	25% 50% 75%			