

# Parallel Interfacing

## Write Cycle

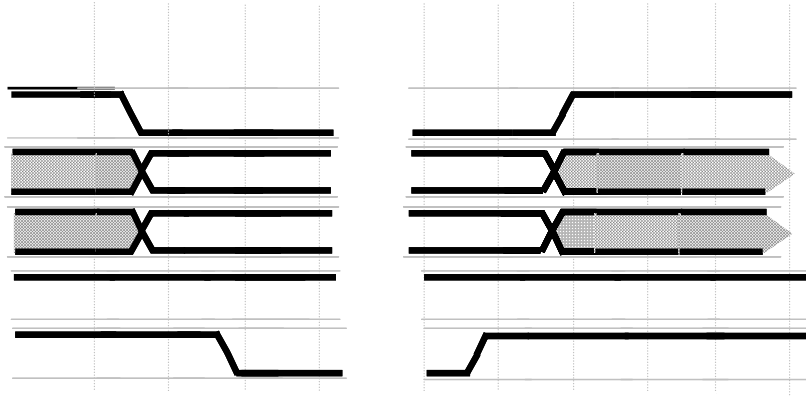


Figure 1: Timing Diagram for a Write Cycle

## Read Cycle

**memory map summary**

Table 3 shows the memory map address ranges of the C6713/13B devices.

**Table 3. TMS320C6713/13B Memory Map Summary**

MEMORY BLOCK DESCRIPTION	BLOCK SIZE (BYTES)	HEX ADDRESS RANGE
Internal RAM (L2)	192K	0000 0000 – 0002 FFFF
Internal RAM/Cache	64K	0003 0000 – 0003 FFFF
Reserved	24M – 256K	0004 0000 – 017F FFFF
External Memory Interface (EMIF) Registers	256K	0180 0000 – 0183 FFFF
L2 Registers	128K	0184 0000 – 0185 FFFF
Reserved	128K	0186 0000 – 0187 FFFF
HPI Registers	256K	0188 0000 – 018B FFFF
McBSP 0 Registers	256K	018C 0000 – 018F FFFF
McBSP 1 Registers	256K	0190 0000 – 0193 FFFF
Timer 0 Registers	256K	0194 0000 – 0197 FFFF
Timer 1 Registers	256K	0198 0000 – 019B FFFF
Interrupt Selector Registers	512	019C 0000 – 019C 01FF
Device Configuration Registers	4	019C 0200 – 019C 0203
Reserved	256K – 516	019C 0204 – 019F FFFF
EDMA RAM and EDMA Registers	256K	01A0 0000 – 01A3 FFFF
Reserved	768K	01A4 0000 – 01AF FFFF
GPIO Registers	16K	01B0 0000 – 01B0 3FFF
Reserved	240K	01B0 4000 – 01B3 FFFF
I2C0 Registers	16K	01B4 0000 – 01B4 3FFF
I2C1 Registers	16K	01B4 4000 – 01B4 7FFF
Reserved	16K	01B4 8000 – 01B4 BFFF
McASP0 Registers	16K	01B4 C000 – 01B4 FFFF
McASP1 Registers	16K	01B5 0000 – 01B5 3FFF
Reserved	160K	01B5 4000 – 01B7 BFFF
PLL Registers	8K	01B7 C000 – 01B7 DFFF
Reserved	264K	01B7 E000 – 01BB FFFF
Emulation Registers	256K	01BC 0000 – 01BF FFFF
Reserved	4M	01C0 0000 – 01FF FFFF
QDMA Registers	52	0200 0000 – 0200 0033
Reserved	16M – 52	0200 0034 – 02FF FFFF
Reserved	720M	0300 0000 – 2FFF FFFF
McBSP0 Data Port	64M	3000 0000 – 33FF FFFF
McBSP1 Data Port	64M	3400 0000 – 37FF FFFF
Reserved	64M	3800 0000 – 3BFF FFFF
McASP0 Data Port	1M	3C00 0000 – 3C0F FFFF
McASP1 Data Port	1M	3C10 0000 – 3C1F FFFF
Reserved	1G + 62M	3C20 0000 – 7FFF FFFF
EMIF CE0 <sup>†</sup>	256M	8000 0000 – 8FFF FFFF
EMIF CE1 <sup>†</sup>	256M	9000 0000 – 9FFF FFFF
EMIF CE2 <sup>†</sup>	256M	A000 0000 – AFFF FFFF
EMIF CE3 <sup>†</sup>	256M	B000 0000 – BFFF FFFF
Reserved	1G	C000 0000 – FFFF FFFF

<sup>†</sup> The number of EMIF address pins (EA[21:2]) limits the maximum addressable memory (SDRAM) to 128MB per CE space.

# LS7266R1

# Encoder to Microprocessor Interface Chip

## Description:

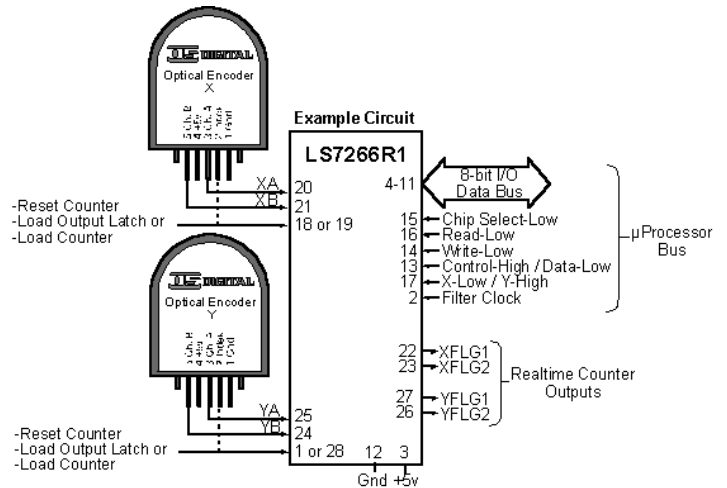
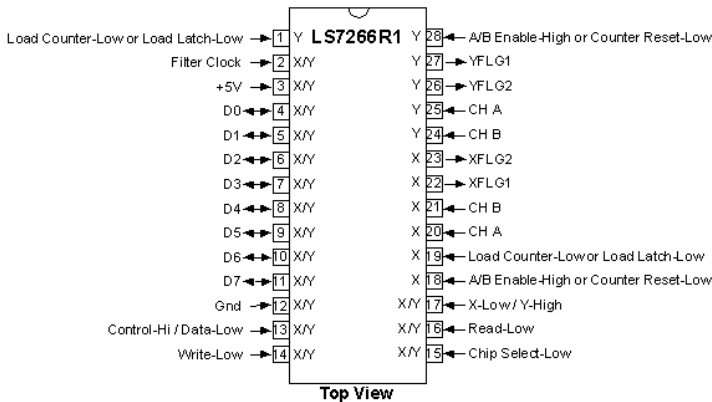
The **LS7266R1** is an LSI monolithic CMOS building block useful in motion control applications. The two 24-bit multimode counters, registers, and logic enables a microprocessor to track the speed, direction, position, and index of one or two optical incremental encoders. In addition to an 8-bit data bus, programmable real-time inputs and outputs are provided for hardware based control functions and status indication.

## Note:

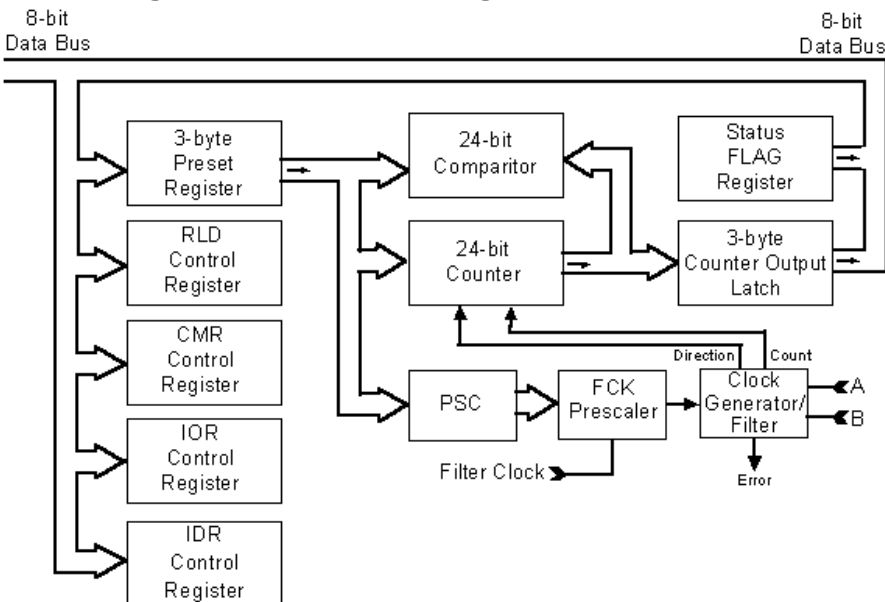
US Digital has already designed the IC's on this data sheet into various products. Please see the **PC7266**, **AD5** or **ED2**.

## Features:

- > X4 or X1 resolution multiplication.
- > Two preloadable 24-bit Up/Down counters.
- > Choice of two 20-pin packages: SOIC surface mount or DIP (600mil).
- > X1 or X2 or X4 resolution multiplier.
- > Binary, BCD, Divide-by-N, Range Limit, Non-Recycle & Non-quadrature Modes.
- > 2-axis 24-Bit comparators.
- > Independent mode programmability for each axis.
- > 17 MHz in quadrature mode.
- > 4 control registers.
- > Readable status flag register.
- > Digital filtering of the input quadrature clocks.
- > Programmable 8-bit separate filter clock prescalers for each axis.
- > Error flags for excess noise.
- > 8-Bit tri-state I/O bus.
- > Latched counter outputs.
- > Input/Output TTL & CMOS compatible.
- > 5 volt operation.



## Block Diagram of Counter & Registers:



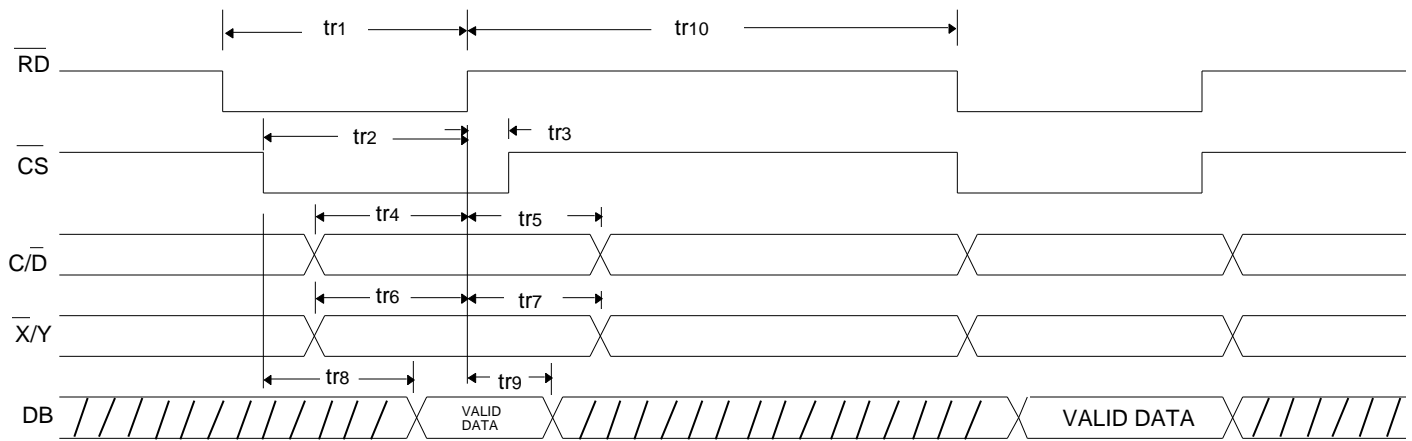
## Ordering Information:

DIP Package (600mil):  
**LS7266R1-DIP**

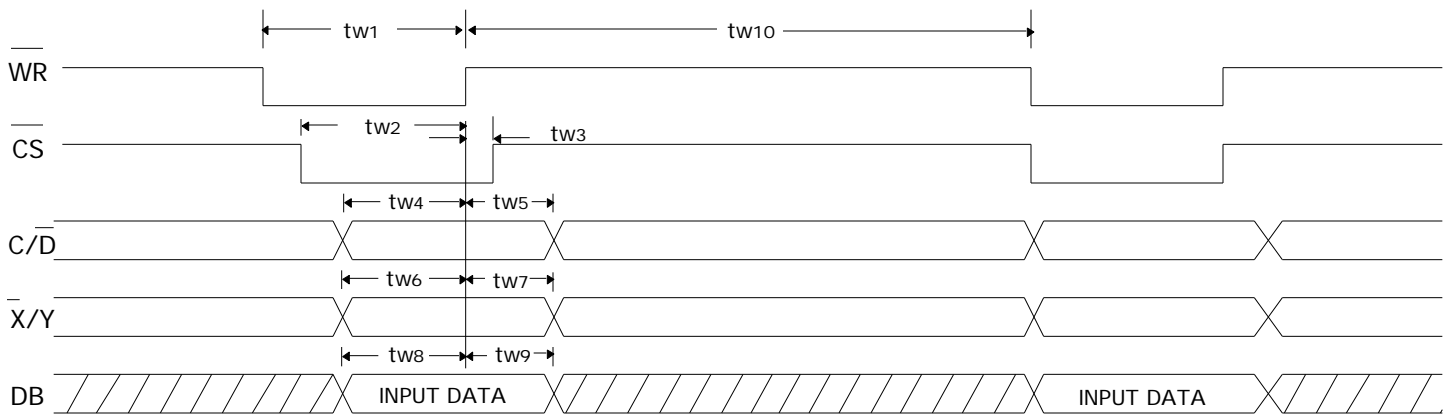
Surface Mount Package:  
**LS7266R1-SOIC**

**Price:**  
\$16.55 / 1  
\$13.25 / 25  
\$10.60 / 100  
\$9.00 / 500  
\$7.65 / 1K

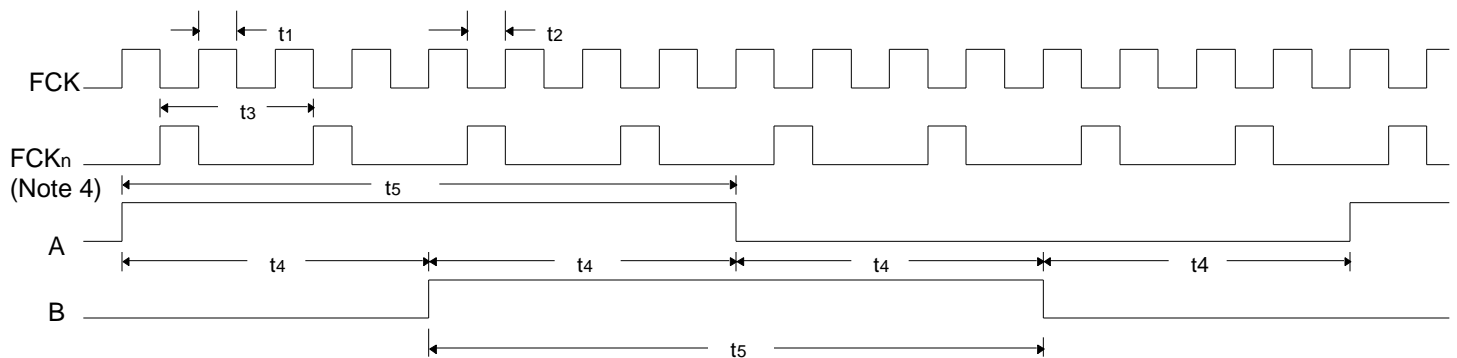
Technical Data, Rev. 11.21.00, November 2000  
All Information subject to change without notice.



**FIGURE 1. READ CYCLE**



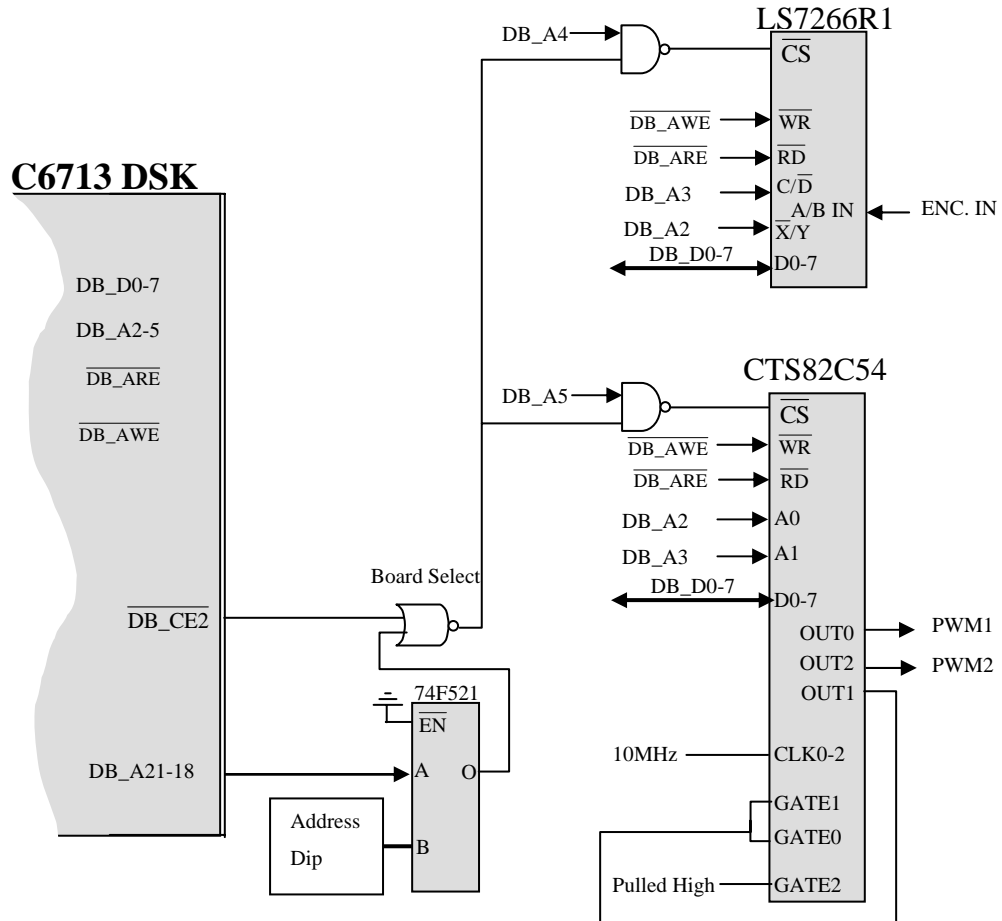
**FIGURE 2. WRITE CYCLE**



**FIGURE 3. FILTER CLOCK FCK AND QUADRATURE CLOCKS A AND B**

**Note 4:**  $FCK_n$  is the final modulo-n internal filter clock, arbitrarily shown here as modulo-1.

# C6713 DSK



```

// Function: void Init_encoders(int chip)
// Parameters:
// chip      1 or 2 indicating which LS7266(encoder) chip to initialize on the daughter card
// Return value: None
// Description: Call this function before using the read_encoder functions. init_encoders enables
// both optical encoder channels on the specified chip. Sets up the chip in X4 quadrature
// decode mode. Zeros the encoder count register.
void Init_encoders(int chip) {

    int addr_offset;

    if (chip == 1) {
        addr_offset = ENC1_OFFSET;
    } else {
        addr_offset = ENC2_OFFSET;
    }

    // Init Index Control Register (IDR) See LS7266 data sheet
    outp(BOARD_ADDR+addr_offset+ENC_CONTROL,ENC_WR_BOTHXY_IDR+ENC_INIT_IDR);

    // Init Input/Output Control Register (IOR) See LS7266 data sheet
    outp(BOARD_ADDR+addr_offset+ENC_CONTROL,ENC_WR_BOTHXY_IOR+ENC_INIT_IOR);

    // Init Counter Mode Register (CMR) See LS7266 data sheet
    outp(BOARD_ADDR+addr_offset+ENC_CONTROL,ENC_WR_BOTHXY_CMR+ENC_INIT_CMR);

    // Init Filter Clock Prescaler Register (PSC) so maximum count frequency is
    // (10MHz/ENC_INIT_PSC)/4. The divide by four is for the X4 quadrature mode.
    writeendr(chip,ENC_INIT_PSC,1);

    // Init Starting Position to zero
    writeendr(chip,0,0);
}

```

```

// Function: void read_encoders(int chip,float *enc1,float *enc2,struct encoder_parameters encpars)
// Parameters:
// chip      1 or 2 indicating which LS7266 chip to read on the daughter card.
// *enc1     A pointer to a variable that will receive the chip's Enc. 1 value in radians.
// *enc2     A pointer to a variable that will receive the chip's Enc. 2 value in radians.
// encpars A structure of type encoder_parameters which contains the needed parameters to
//          convert the channel's encoder count to radians.
//          See c6xdskdigio.h for a description of encoder_parameters.
// Return value:      None
// Description: This function reads both encoder channels of the specified chip and returns the
// values in radians with the pointers enc1 and enc2.
void read_encoders(int chip,float *enc1,float *enc2,struct encoder_parameters encpars) {
    int data;
    int data0;
    int data1;
    int data2;
    int counter, curr_counter;
    int addr_offset;

    // Determine which LS7266 chip to read.
    if (chip == 1) { // Encoder Channels 1 and 2
        addr_offset = ENC1_OFFSET;
    } else { // Encoder Channels 3 and 4
        addr_offset = ENC2_OFFSET;
    }
    //Transfer CNTR (Counter) to OL (Output Latch), Reset BP(Byte Pointers) on all counters
    //See LS7266 data sheet
    outp(BOARD_ADDR+addr_offset+ENC_CONTROL, ENC_WR_BOTHXY_RLD+ENC_CNTR_TO_OL);

    // Read from both counters
    for (counter=0; counter <2; counter++) {
        // Select Counter
        switch (counter) {
            case 0: curr_counter = addr_offset+ENC_DATA+ENC_X;
                    break;
            case 1: curr_counter = addr_offset+ENC_DATA+ENC_Y;
                    break;
        }
        // read 24 bit counter value
        data0 = inp(BOARD_ADDR+curr_counter);
        data1 = inp(BOARD_ADDR+curr_counter);
        data2 = inp(BOARD_ADDR+curr_counter);

        data0 = data0 & ENC_EIGHT_LSB;
        data1 = data1 & ENC_EIGHT_LSB;
        data2 = data2 & ENC_EIGHT_LSB;

        // Build 24 bit integer from 3 8bit values
        data1 = data1 << 8;
        data2 = data2 << 16;
        data = data2 | data1 | data0;

        // Convert 24 bit unsigned integer to signed 24 bit integer
        // this will cause the 24 bit count to count up to (2^24 - 1)/2, and down to -(2^24)/2
        if (data >= 8388608)
            data = data - 16777216;

        //Convert signed integer counts to float radians.
        // This uses the encoder_parameters structure "encpars" passed parameter to
        // perform the conversion to radians.
        // See c6xdskdigio.h for a discription of the encoder_parameters sturcture
        switch (counter) {
            case 0: *enc1 = (encpars.enc1_polarity*data)*((2*PI)/encpars.enc1_cnts_per_rev)+encpars.enc1_init_rad_coordinates;
                    break;
            case 1: *enc2 = (encpars.enc2_polarity*data)*((2*PI)/encpars.enc2_cnts_per_rev)+encpars.enc2_init_rad_coordinates;
                    break;
        } // end switch
    } // endfor
}

```

```

// *****
// Definitions for operating C6XDSK Daughter Card
// *****
#define HALFPI 1.5707963
#define G 9.804 // acceleration of gravity m/s^2

#define EMIF_CE2      0x1800010 // Address of EMIF CE2 control register
#define EMIF_CE3      0x1800014 // Address of EMIF CE3 control register

// C6XDSKDIGIO daughter card base address (J1 ON ON ON ON)
#define BOARD_ADDR 0xA000000

// Defines for LS7266 interface
#define ENC1_OFFSET      0x10
#define ENC2_OFFSET      0x40
#define ENC_X             0x0
#define ENC_Y             0x4
#define ENC_DATA         0x0
#define ENC_CONTROL      0x8
#define ENC_WR_RLD       0x00
#define ENC_WR_BOTHXY_RLD 0x80
#define ENC_WR_CMR       0x20
#define ENC_WR_BOTHXY_CMR 0xa0
#define ENC_WR_IOR       0x40
#define ENC_WR_BOTHXY_IOR 0xc0
#define ENC_WR_IDR       0x60
#define ENC_WR_BOTHXY_IDR 0xe0
#define ENC_INIT_CMR     0x18
#define ENC_INIT_IOR     0x01
#define ENC_INIT_IDR     0x00
#define ENC_INIT_PSC     0x0a
// Settings for both RLD commands
#define ENC_PR_TO_CNTR   0x08
#define ENC_PR_TO_PSC   0x18
#define ENC_CNTR_TO_OL   0x11
#define ENC_RESET_BP    0x01
// Masking bytes - AND
#define ENC_EIGHT_LSB   0xFF
#define ENC_EIGHT_SB    0xFF00
#define ENC_EIGHT_MSB    0xFF0000

// definition of the inp and outp functions. inp and outp are simply memory map accesses
// I use the inp and outp functions when communicating with the daughter card to emphasize that
// the daughter card is an I/O peripheral board.
#define inp(a)          *((unsigned volatile int *) (a))
#define outp(a,b)       *((unsigned volatile int *) (a)) = (b)

```

## Chip Access:

D7	D6	D5	C/D	RD	WR	X/Y	CS	Function
X	X	X	X	X	X	X	1	Disable Chip
0	0	0	1	1	$\overline{\text{H}}$	0	0	Write to XRLD
0	0	0	1	1	$\overline{\text{L}}$	0	0	Write to YRLD
1	0	0	1	1	$\overline{\text{H}}$	X	0	Write to both XRLD and YRLD
0	0	1	1	1	$\overline{\text{H}}$	0	0	Write to XCMR
0	0	1	1	1	$\overline{\text{L}}$	0	0	Write to YCMR
1	0	1	1	1	$\overline{\text{H}}$	X	0	Write to both XCMR and YCMR
0	1	0	1	1	$\overline{\text{H}}$	0	0	Write to XIOR
0	1	0	1	1	$\overline{\text{L}}$	0	0	Write to YIOR
1	1	0	1	1	$\overline{\text{H}}$	X	0	Write to both XIOR and YIOR
0	1	1	1	1	$\overline{\text{H}}$	0	0	Write to XIDR
0	1	1	1	1	$\overline{\text{L}}$	0	0	Write to YIDR
1	1	1	1	1	$\overline{\text{H}}$	X	0	Write to both XIDR and YIDR
X	X	X	0	1	$\overline{\text{H}}$	0	0	Write to X Preset Register, increment Address Counter
X	X	X	0	1	$\overline{\text{L}}$	0	0	Write to Y Preset Register, increment Address Counter
X	X	X	0	$\overline{\text{H}}$	1	0	0	Read X Output Latch, increment Address Counter
X	X	X	0	$\overline{\text{L}}$	1	0	0	Read Y Output Latch, increment Address Counter
X	X	X	1	$\overline{\text{H}}$	1	0	0	Read X FLAG Register
X	X	X	1	$\overline{\text{L}}$	1	0	0	Read Y FLAG Register

**Writing to 1 of the 4 Control Registers:** Set Control/Data high. Bits 5 and 6 are used as address bits to select one of the 4 registers. Bit 7 allows the data to apply to both X and Y registers and overrides X/Y input. Only bits 0-4 are stored.

### Notes:

- 1) D7 is the Most significant bit of the data bus.
- 2) X means "don't care".

## Output Latch (Read Only, Data):

The 24-bit counter value at any instant can be accessed by transferring its contents to the 24-bit Output Latch. Note that only good stable data will be passed from the counter to the Output Latch even if the counter bits are in the midst of a transition. This chip will internally stretch the latch pulse if necessary until the counter has stabilized. The 3 bytes are then read from the Output Latch (least significant byte 1st). The byte pointer is automatically incremented with each read cycle. You must reset the byte pointer (BP) before making the first read.

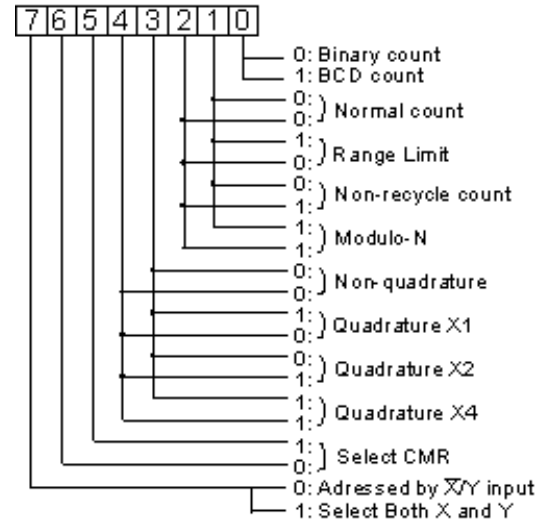
## Preset Register (Write Only, Data):

The 24-bit preset register is the input port for the 24-bit counter and the filter clock prescaler (PSC). The data is first written into the preset register in 3 write cycles (least significant byte 1st). The byte pointer is automatically incremented with each write cycle. You must reset the byte pointer (BP) before making the first write.

## Filter Clock Prescalers (XPSC & YPSC):

Each prescaler (PSC) is an 8-bit programmable modulo-N down counter, driven by the filter clock input (FCK). The factor N is loaded into a PSC, from the preset register (PR) byte 0, in the RLD register. This allows the ability to generate independent filter clock frequencies for each channel. Final filter clock frequency = (FCK / (PSC + 1)).

## Counter Mode Register (CMR):



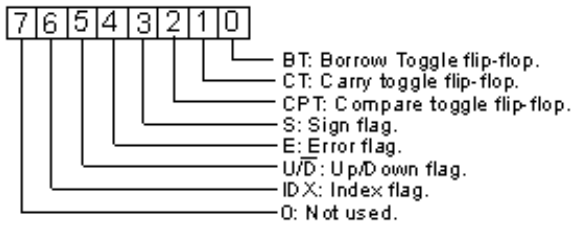
**Range Limit:** In range limit count mode, an upper and a lower limit is set, mimicking limit switches in the mechanical counterpart. The upper limit is set by the content of the PR and the lower limit is set to be 0. The CNTR freezes as CNTR = PR when counting up and at CNTR = 0 when counting down. At either of these limits, the counting is resumed only when the count direction is reversed.

**Non-Recycle:** In non-recycle count mode the CNTR is disabled, whenever a count overflow or underflow takes place. The end of cycle is marked by the generation of a Carry (in Up Count) or a borrow (in Down Count). The CNTR is re-enabled when a reset or load operation is performed on the CNTR.

**Modulo-N:** In modulo-N count mode, a count boundary is set between 0 and the content of the PR. When counting up, at CNTR = PR, the CNTR is reset to 0 and the up count is continued from that point. When counting down, at CNTR = 0, the CNTR is loaded with the content of PR and down count is continued from that point.

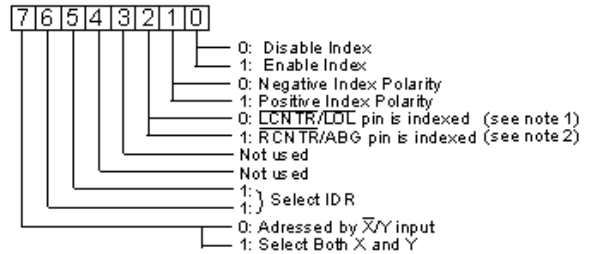
The modulo-N is true bidirectional in that the divide-by-N output frequency is generated in both up and down direction of counting for same N and does not require the complement of N in the UP instance. In frequency divider application the modulo-N output frequency can be obtained at either the compare (CY) or the BW output. Modulo-N output frequency  $f_N = (f_i / (N+1))$  where  $f_i$  = Input count frequency and  $N = PR$ .

## Status FLAG Register (Read only, control):



**BT:** Toggles every time CNTR underflows  
**CT:** Toggles every time CNTR overflows  
**CPT:** Toggles every time PR = CNTR  
**S:** Reset to 0 when CNTR overflows  
**E:** Set to 1 when excessive noise is present at the count inputs.  
**U/D:** Set to 1 when counting up and reset to 0 when counting down.  
**IDX:** Set to 1 when index is valid.  
 The FLAG registers hold the status information of the CNTRs and can be read out on the data bus when C/D = 1. E = 1 indicates excessive noise at the inputs but not a definite count error. Once set, E can only be reset via the RLD.

## Index Control Register (IDR):

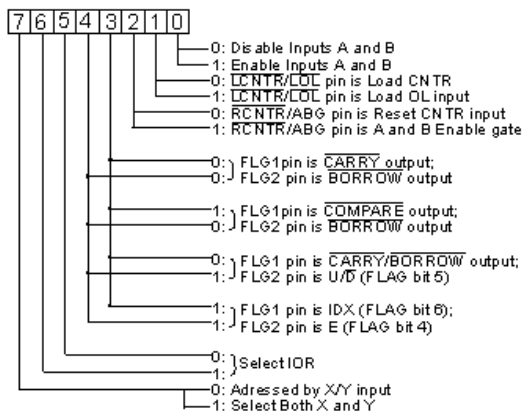


**Note 1:**  $\overline{\text{LCNTR/LOL}}$  input must also be initialized as the Load CNTR or the Load OL input via the IOR register bit 1.

**Note 2:**  $\overline{\text{RCNTR/ABG}}$  input must also be initialized as the reset CNTR input via the IOR register bit 2.

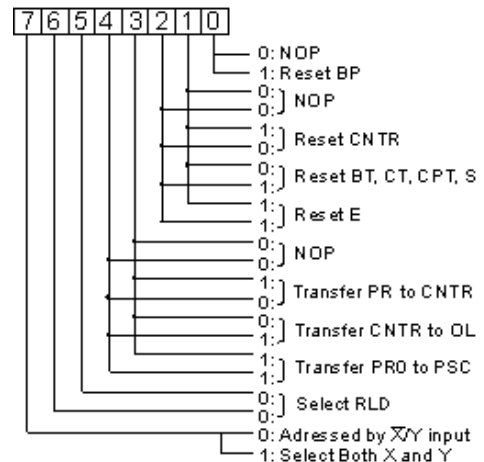
Either the  $\overline{\text{LCNTR/LOL}}$  or the  $\overline{\text{RCNTR/ABG}}$  inputs can be initialized to operate as an index input. When initialized as such, the index signal from the encoder, applied to one of these inputs performs either the Reset CNTR or the Load CNTR or the Load OL operation synchronously with the quadrature clocks. Note that only one of these inputs can be selected as the Index at a time and hence only one type of indexing function can be performed in any given setup. The index function must be disabled in non-quadrature count mode.

## Input/Output Control Register (IOR):



Control functions may be combined. The toggle flip flops are triggered by the trailing edges of the associated Carry, Borrow, or Compare match. Thus there is a 1-clock delay between the input and output of each flip flop. Unless otherwise specified, assume the longest prop delay from any input to any output is <110ns.

## Reset & Load Signal Decoders (RLD):



## Absolute Maximum Ratings:

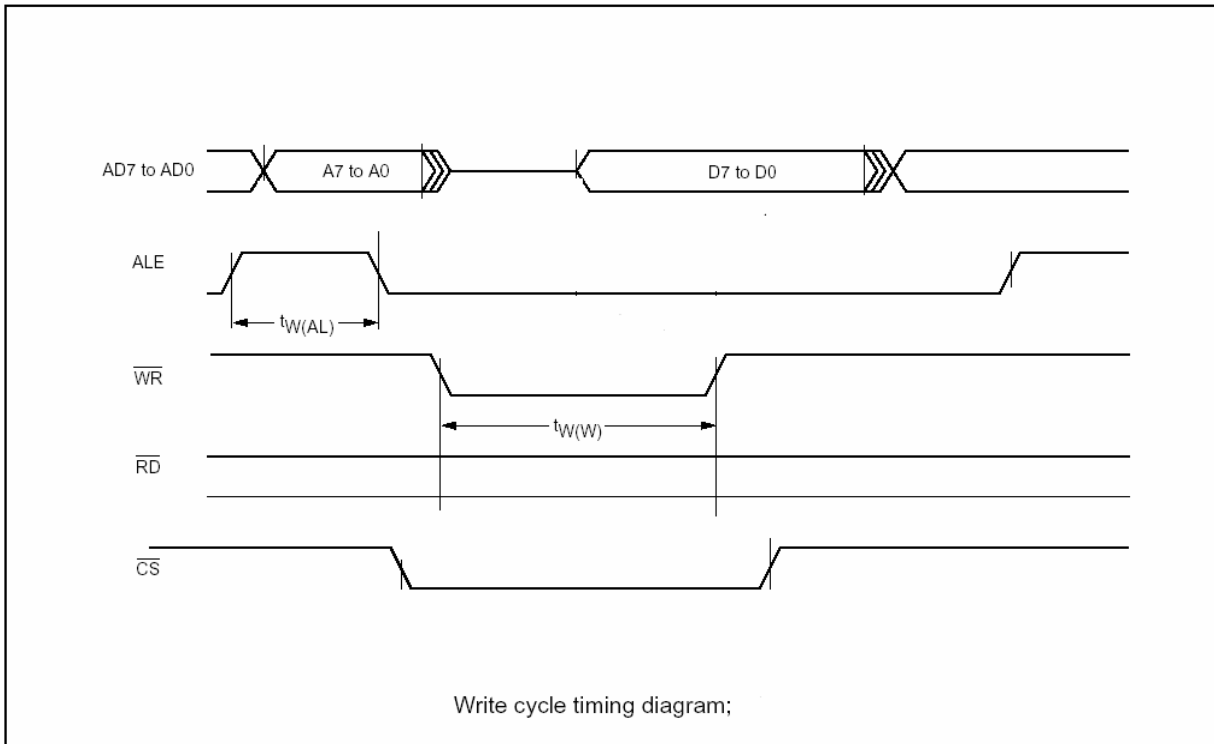
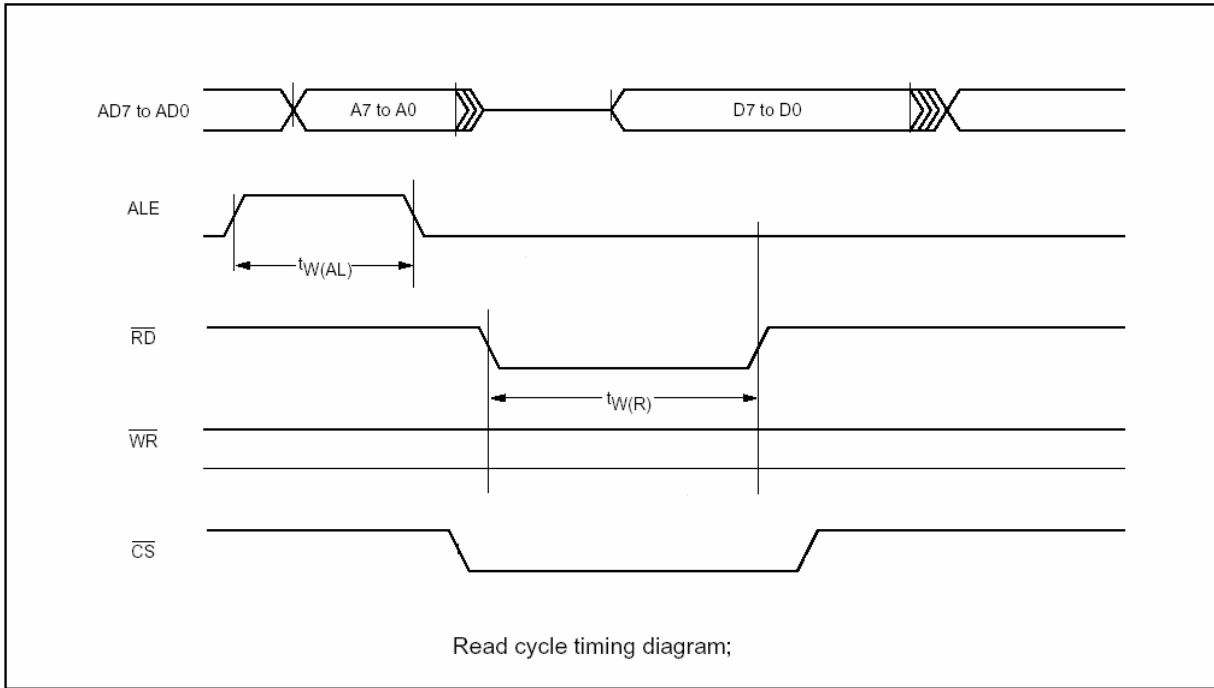
Parameter	Min.	Max.	Units
Voltage at any input	-.5	VCC+.5	Volts
Supply voltage (VCC)	-	7	Volts
Operating temperature	-25	80	°C
Storage temperature	-65	150	°C

## DC Electrical Characteristics:

Parameter	Min.	Max.	Units	Notes
Supply voltage	4.5	5.5	Volts	
Supply current		800	µA	all clocks off
Input logic low		0.8	Volts	
Input logic high	2.0		Volts	
Output low voltage	0.5		Volts	I <sub>OutSink</sub> =5mA
Output high voltage	VCC-.5		Volts	I <sub>OutSource</sub> =1mA
Input leakage current		30	nA	
Output source current	1		mA	V <sub>O</sub> = VCC-.5V
Output sink current	5		mA	V <sub>O</sub> = 0.5V
Data bus leakage		60	nA	data bus off current

# Clarification of DS17887 Read and Write Cycles.

## AC timing diagrams



To design a low pass or a high pass filter we will use the “fir1” function in Matlab. “B” returned from “fir1” are the coefficients of the filter

Wn is the nyquist frequency which is half of your sample frequency.

```
>> help fir1
```

FIR1 FIR filter design using the window method.

B = FIR1(N,Wn) designs an N'th order lowpass FIR digital filter and returns the filter coefficients in length N+1 vector B.

The cut-off frequency Wn must be between  $0 < Wn < 1.0$ , with 1.0 corresponding to half the sample rate. The filter B is real and has linear phase. The normalized gain of the filter at Wn is -6 dB.

B = FIR1(N,Wn,'high') designs an N'th order highpass filter.

To remember the form of the discrete equations to implement inside your periodic function type “help filter”. We are not using this function but it shows us the order of the B coefficients. Note that in the case of a FIR filter  $a(1) = 1$  and all other “a” coefficients are 0.

```
>> help filter
```

FILTER One-dimensional digital filter.

Y = FILTER(B,A,X) filters the data in vector X with the filter described by vectors A and B to create the filtered data Y. The filter is a "Direct Form II Transposed" implementation of the standard difference equation:

$$a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + \dots + b(nb+1)*x(n-nb) \\ - a(2)*y(n-1) - \dots - a(na+1)*y(n-na)$$

So in C we need to save past states to implement this filter. The number of past states is determined by the order of the filter designed.

To Cut and Paste your filter coefficients from Matlab to C using the command:

**arraytoCformat(b')**