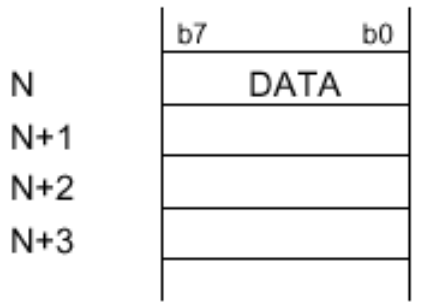

Microcontroller Introduction

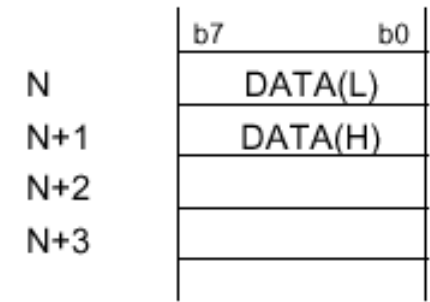
Data Formats for the Renesas Microcontroller

Byte

- 8 bits
- signed & unsigned
- unsigned range
0 to 255
- unsigned char a;



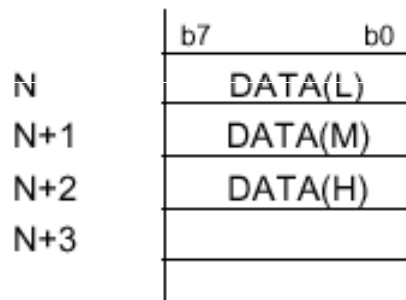
Byte (8-bit) data



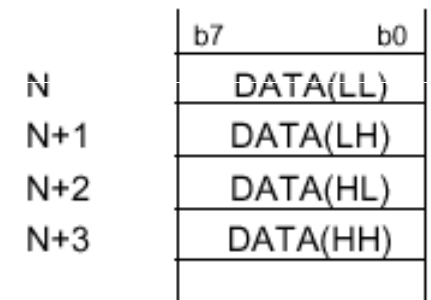
Word (16-bit) data

Word

- 16 bits
- signed & unsigned
- unsigned range
0 to 65535
- int b; unsigned int b;



20-bit (Address) data



Long Word (32-bit) data

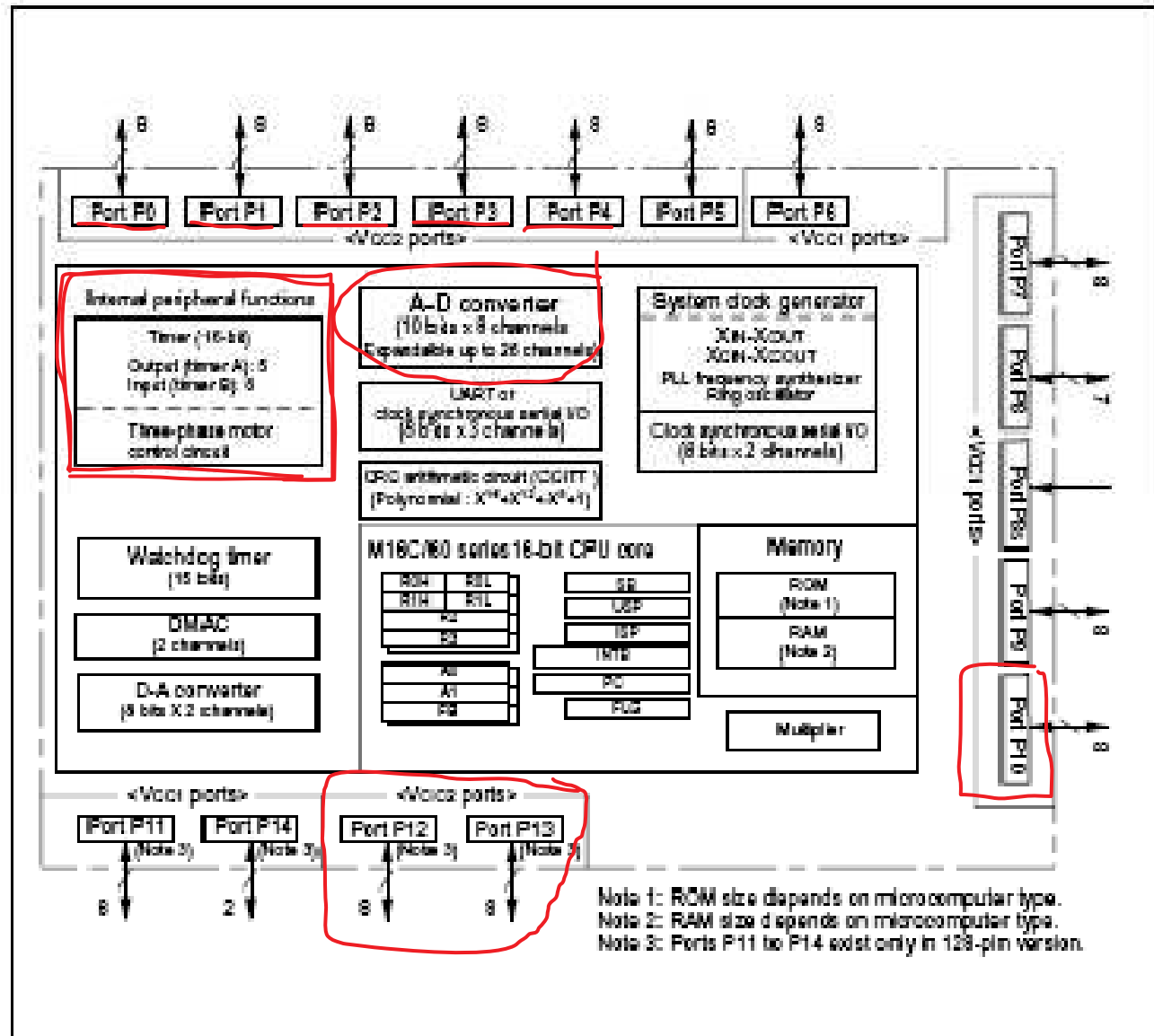
Longword

- 32 bits, signed and unsigned, unsigned range 4,294,967,294
- long int c; unsigned long int c;

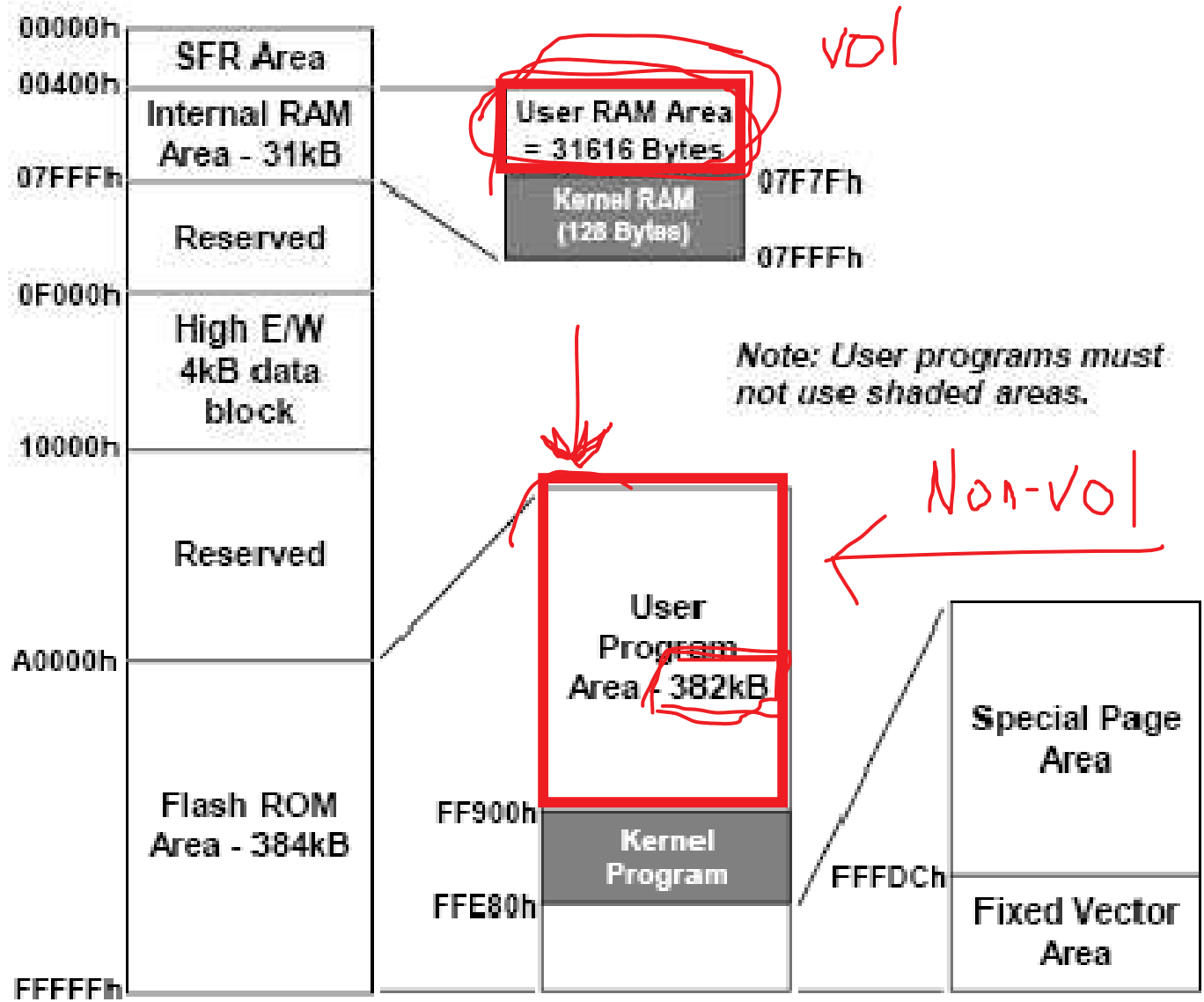
Review of the Renesas Architecture

Microcontroller has:

- General Purpose Registers
- RAM
- Flash
- EEPROM
- Digital Ports
- Analog Ports
- Timers
- Oscillator
- DMA Controller
- Reliability and safety



Memory Map for Renesas Microcontroller

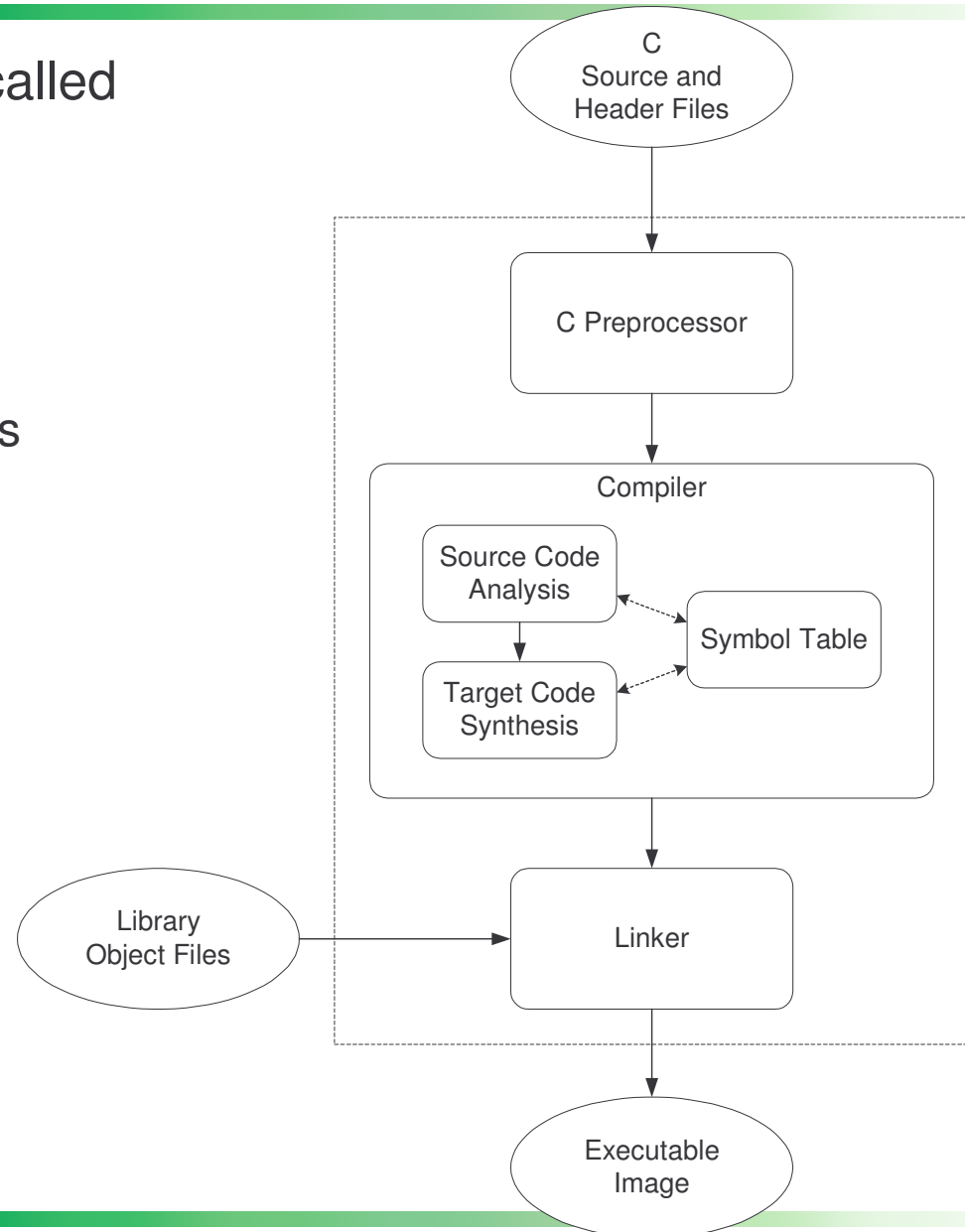


A C Code “Project”

- You will use an “Integrated Development Environment” (IDE) to develop, compile, load, and debug your code.
- Your entire code package is called a *project*. Often you create several files to split the functionality:
 - Several C files
 - Several include (.h) files
 - Maybe some assembly language (.a30) files
 - Maybe some assembly language include (.inc) files
- A lab, like “Lab7”, will be your project. You may have three .c, three .h, one .a30, and one .inc files.

Compiling a C Program

- Entire mechanism is usually called the “compiler”
- **Preprocessor**
 - macro substitution
 - conditional compilation
 - “source-level” transformations
 - output is still C
- **Compiler**
 - generates object file
 - machine instructions
- **Linker**
 - combine object files (including libraries) into executable image



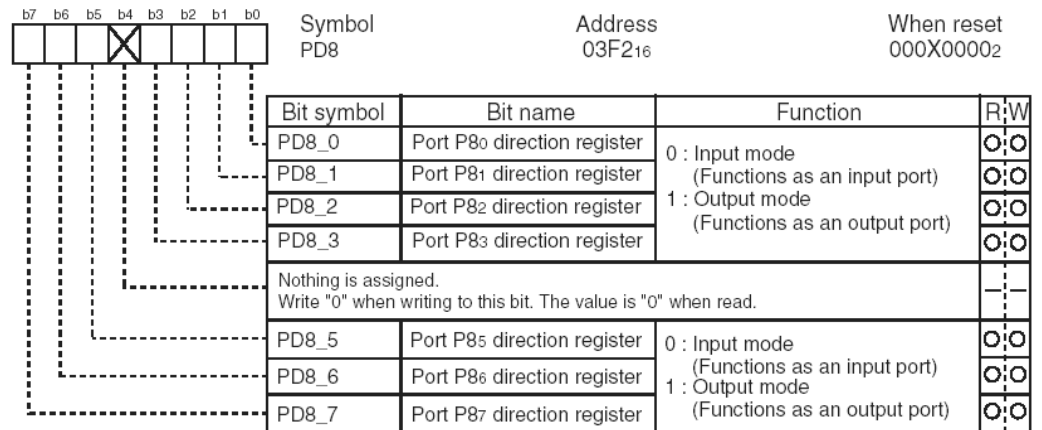
Compiler

- **Source Code Analysis**
 - “front end”
 - parses programs to identify its pieces
 - variables, expressions, statements, functions, etc.
 - depends on language (not on target machine)
- **Code Generation**
 - “back end”
 - generates machine code from analyzed source
 - may optimize machine code to make it run more efficiently
 - very dependent on target machine
- **Symbol Table**
 - map between symbolic names and items
 - like assembler, but more kinds of information

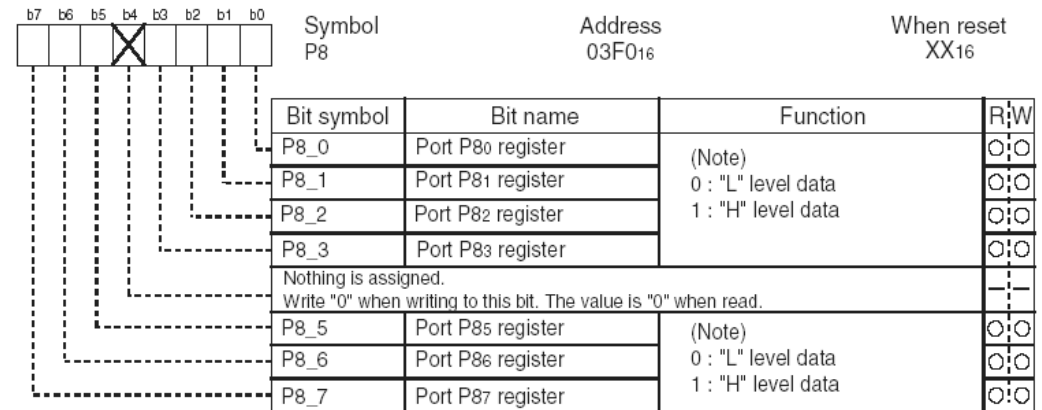
Digital Input/Output (I/O) Ports

- The fundamental interfacing subsystem
 - Port bits can be inputs or outputs
 - M30626 has fourteen *Programmable I/O Ports* (total of 106 digital I/O bits) (P0 to P13 = 8 bits each, P14 = 2)
 - For some other MCUs some ports may be limited to only input or output
- Direction register sets bit direction
 - Port Direction register names PDx
 - 1: Output
 - 0: Input
- Data register holds actual data
 - Port data register names: Px
- M16C62P Hardware Manual*

Port P8 direction register

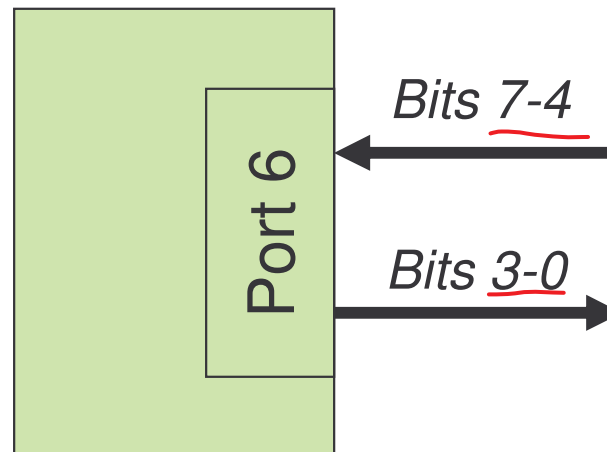


Port P8 register



Example: P6 Echoes Nibble Data

- Configuring port to desired structure
 - Top 4 bits (4-7) of P6 are inputs
 - Clear bits 4-7 of PD6
 - Bottom 4 bits of P6 are outputs
 - Set bits 0-3 of PD6



Example in C: P6 Echoes Nibble Data

- Reading and writing data
 - Load data from input port
 - Move top nibble to bottom
 - Write data to output port
 - Jump back to start
- Now let's invert the data before writing it out
 - Load data from input port
 - Move top nibble to bottom
 - Invert it (complement)
 - Write data to output port
 - Jump back to start

```
#include "sfr62p.h"
#define DIR_OUT (1)
#define DIR_IN (0)
unsigned char a;

/* pd6 = 0xf0; */
pd6_0 = pd6_1 = DIR_OUT;
pd6_2 = pd6_3 = DIR_OUT;
pd6_4 = pd6_5 = DIR_IN;
pd6_6 = pd6_7 = DIR_IN;
while (1) {
    a = p6;
    a >>= 4;
    p6 = a;
}
```

Sample Code from Demo

```
#include "stdio.h" /* sprintf */
#include "sfr62p.h"
#include "SKP_LCD.h"
#include "string.h"

#define RED_LED (p8_0) /* from board schem.*/
#define YEL_LED (p7_4)
#define GRN_LED (p7_2)
#define LED_ON (0) /* 0 is ON for LEDs */
#define LED_OFF (1)

#define DIR_IN (0)
#define DIR_OUT (1)

#define SW1 (p8_3)
#define SW2 (p8_2)
#define SW3 (p8_1)

void init_switches() {
    pd8_1 = pd8_2 = pd8_3 = DIR_IN;
}

void init_LEDs() {
    pd8_0 = pd7_4 = pd7_2 = DIR_OUT;
    RED_LED = YEL_LED = GRN_LED = LED_ON;
    RED_LED = YEL_LED = GRN_LED = LED_OFF;
}

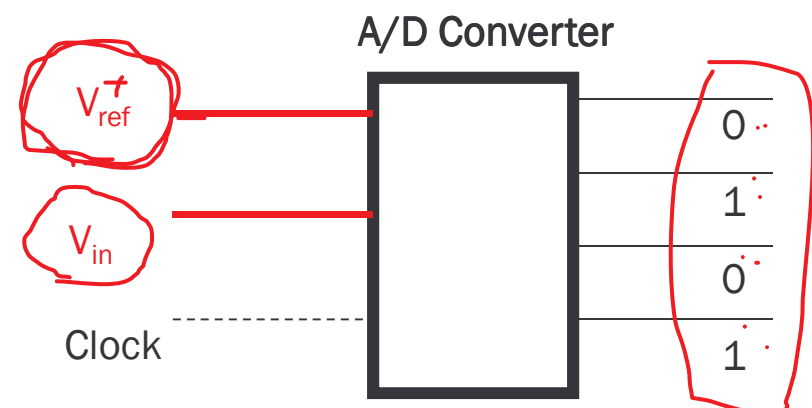
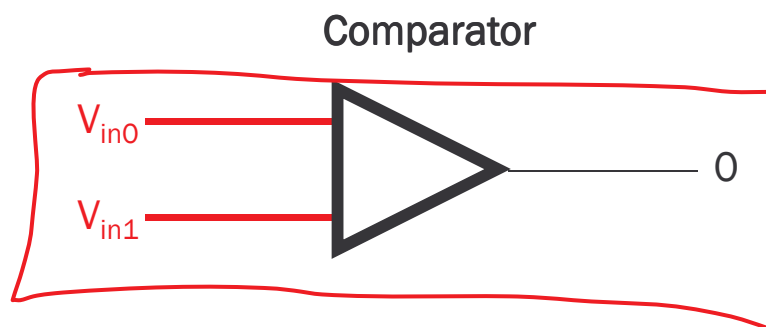
void test_switches(void) {
    while (1) {
        RED_LED = (!SW1)? LED_ON : LED_OFF;
        YEL_LED = (!SW2)? LED_ON : LED_OFF;
        GRN_LED = (!SW3)? LED_ON : LED_OFF;
    }
}

void main () {
    char buf[9];
    long int i, r=12345;

    init_switches();
    init_LEDs();
    InitDisplay();
    #if (0)
        test_switches();
    #endif
    DisplayString(LCD_LINE1, "Response");
    DisplayString(LCD_LINE2, " Timer ");
    while(1) {
        for (i=0; i<200000+(r%50000); i++)
            ;
        i=0;
        RED_LED = YEL_LED = GRN_LED = LED_ON;
        while (SW1)
            i++;
    #if (1)
        sprintf(buf, "%8ld", i);
        DisplayString(LCD_LINE1, buf);
        DisplayString(LCD_LINE2, "iters. ");
    #else
        sprintf(buf, "%8.3f", i*39.1/287674);
        DisplayString(LCD_LINE1, buf);
        DisplayString(LCD_LINE2, "millisec");
    #endif
        RED_LED = YEL_LED = GRN_LED = LED_OFF;
        r=0;
        while (!SW1) /* wait for switch to come
up */
            r++;
    }
}
```

From Analog to Digital

- Embedded systems often need to measure values of physical parameters
- These parameters are usually continuous (*analog*) and not in a digital form which computers (which operate on discrete data values) can process
- A **Comparator** is a circuit which compares an analog input voltage with a reference voltage and determines which is larger, returning a 1-bit number
- An **Analog to Digital converter** [AD or ADC] is a circuit which accepts an analog input signal (usually a voltage) and produces a corresponding multi-bit number at the output.



ADC Basic Functionality

2.5
5.0v
0v
4

- n = converted code
- V_{in} = sampled input voltage
- V_{+ref} = upper end of input voltage range
- V_{-ref} = lower end of input voltage range
- N = number of bits of resolution in ADC

$$n = \left[\frac{(V_{in} - V_{-ref})(2^N - 1)}{V_{+ref} - V_{-ref}} + 1/2 \right]_{int}$$

$$n = \left[\frac{(V_{in})(2^N - 1)}{V_{+ref}} + 1/2 \right]_{int}$$

$$n = \left[\frac{3.30v(2^{10} - 1)}{5v} + 1/2 \right]_{int} = 675$$

$$n = \left[\frac{2.5 \times 15}{5} + .5 \right]$$

$$n = 8$$

$$= 1000$$

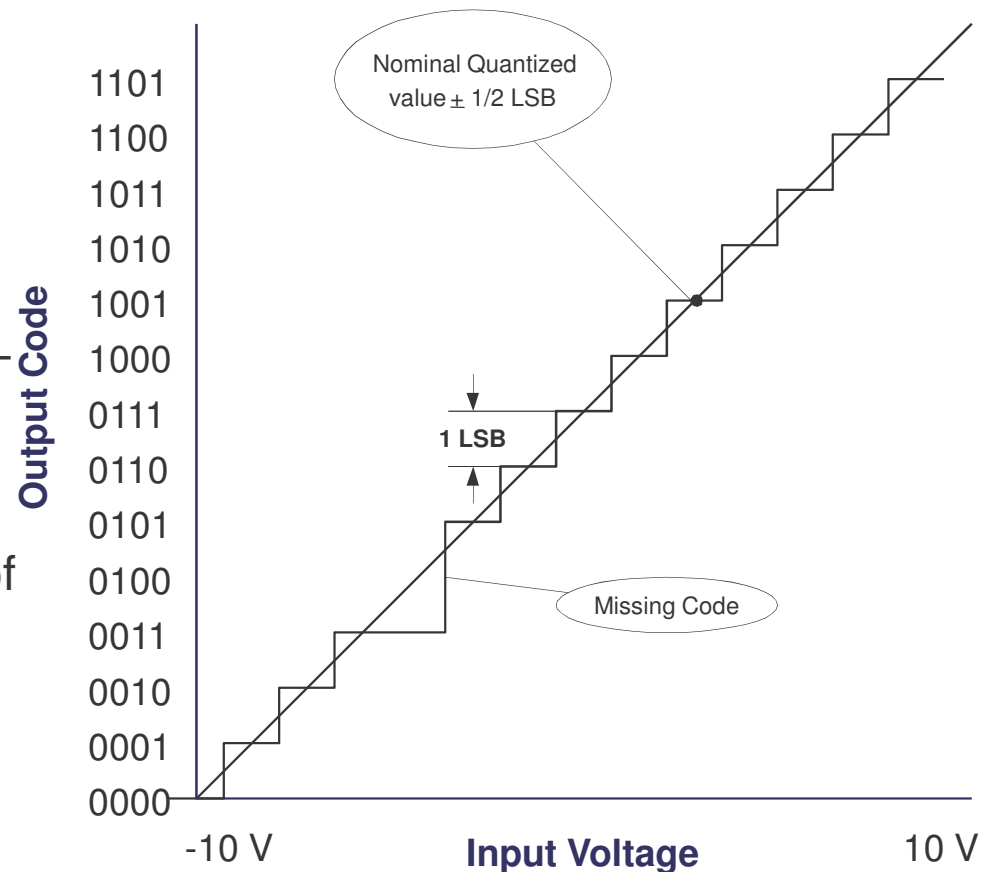
if $V_{-ref} = 0v$

$$2^{10} = 1024$$

ADC Transfer Function

•The ideal output from an A/D converter is a stair-step function (see right)

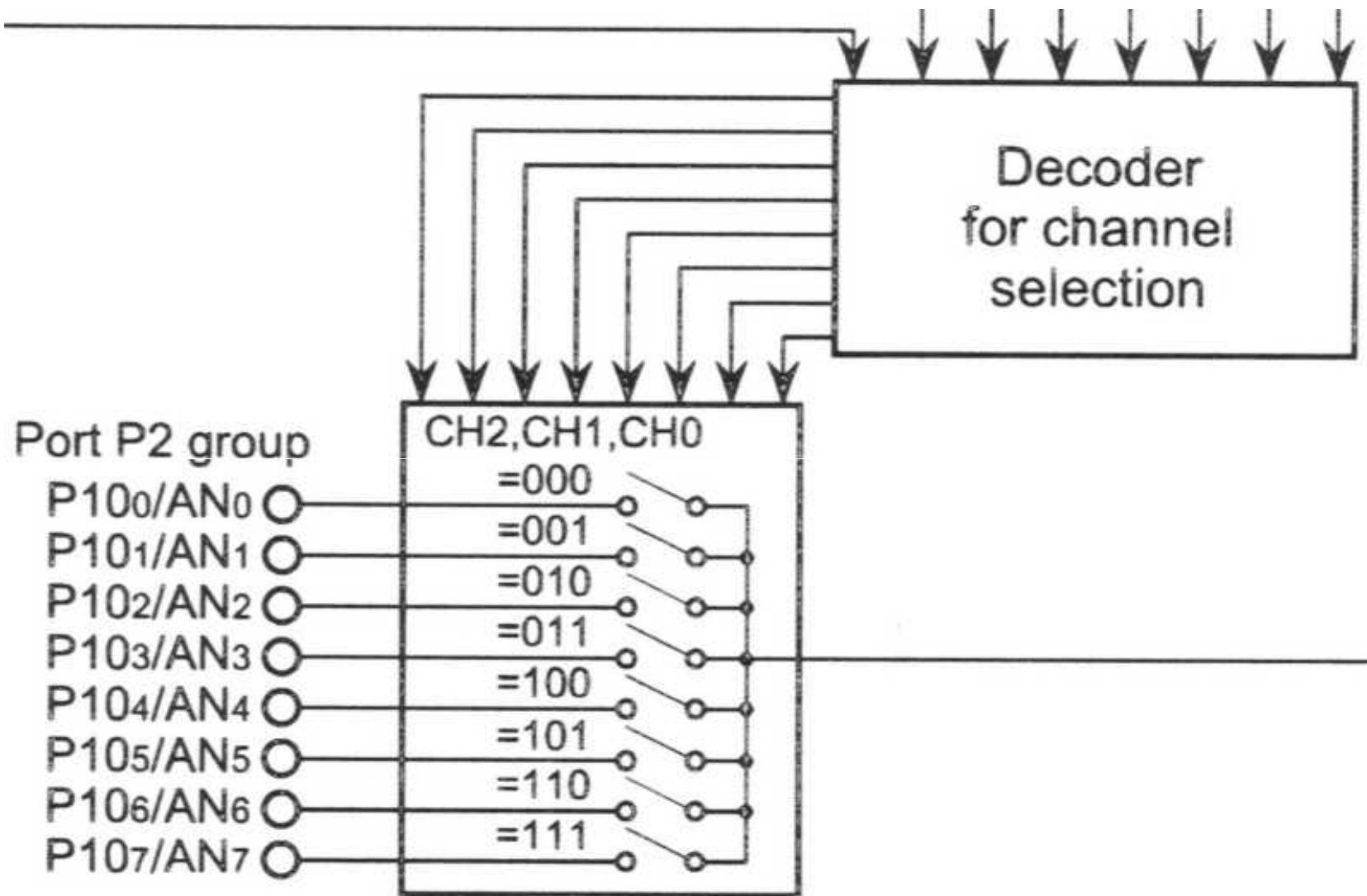
- Ideal worst case error in conversion is $\pm 1/2$ bit.
- Missing codes or the imperfections where increasing voltage does not result in the next step being output are described as non-monotonicity.
- Errors in A/D conversion may be significant particularly if the full range of the analog signal is significantly less than the range of the analog input of the A/D.



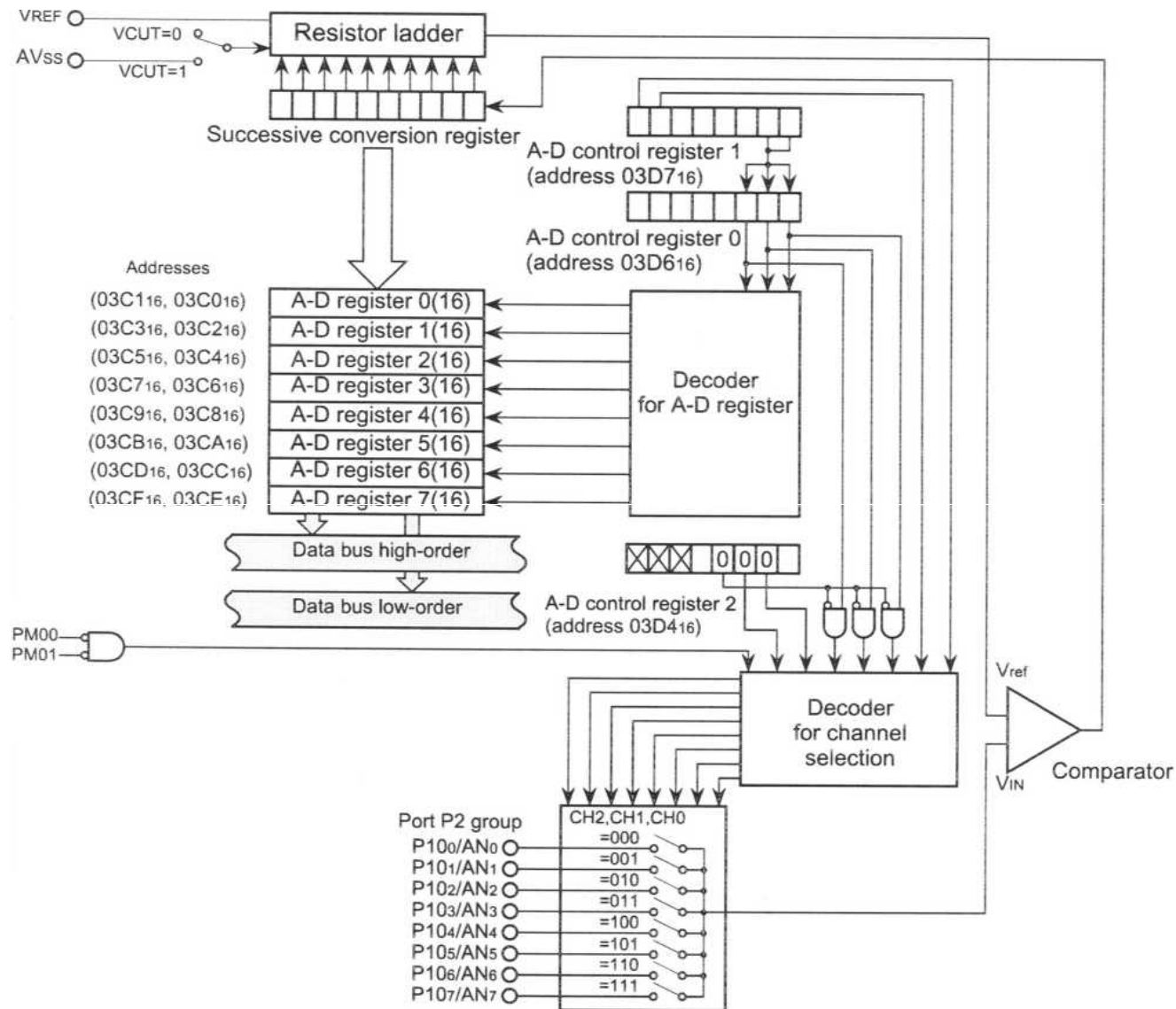
M30626P ADC Peripheral

- 10 bit successive approximation converter, can operate in 8 bit mode
- Input voltage: 0 to V_{CC}^5
- Reference voltage applied to V_{REF} pin
 - Can be disconnected with VCUT bit to save power
- Input Multiplexer: 8 input channels

Input Mux (262, but 626 similar)



M30262 Converter Overview (626P similar)



Repeated ADC

- The microcontroller performs repeated A/D conversions, and can read data whenever needed

```
adcon0 = 0x88;
```

```
adcon1 = 0x28;
```

```
adcon2 = 0x01;
```

```
adst = 1; // Start a conversion here
```

Then in your procedure

```
TempStore = ad0 & 0x03ff;
```