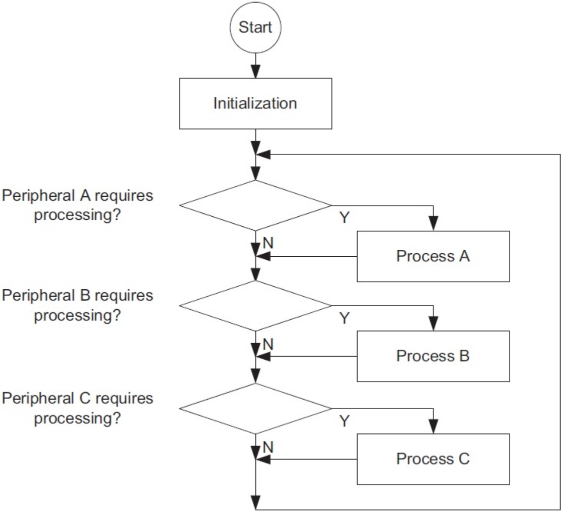
Embedded C Programming with 8051

Sep,25, 2021

Embedded Software Program flows

- There are many different ways to structure the flow of the application program.
- Polling or Superloop Easy to develop, works well for simple tasks.
- Interrupt Driven Works well for low power applications.
- Combination of Interrupt and Polling task can be divided into ISR and process.
- Breaking a task into a sequence of states Each time one state of the process is executed.
- Using an RTOS An operating system manages multiple tasks.

Polling or Superloop



Polling or Superloop

```
void main(void)
    {
    // Prepare run function X
    X_Init();

    while(1) // 'for ever' (Super Loop)
        {
        X(); // Run function X()
        }
    }
```

Figure: Superloop

Interrupt driven Application

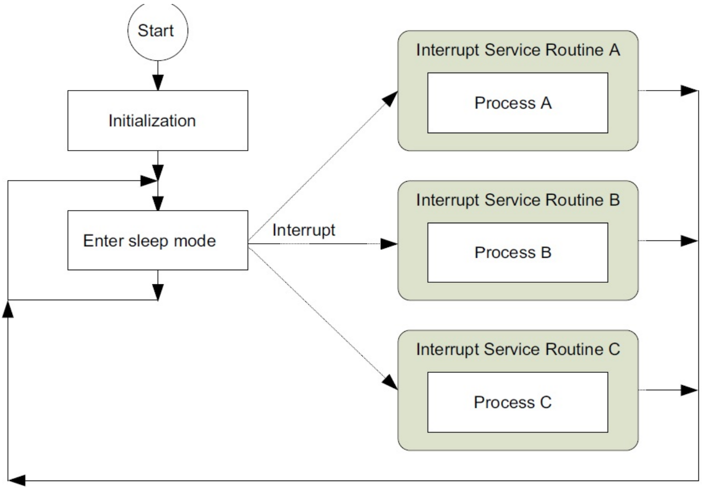
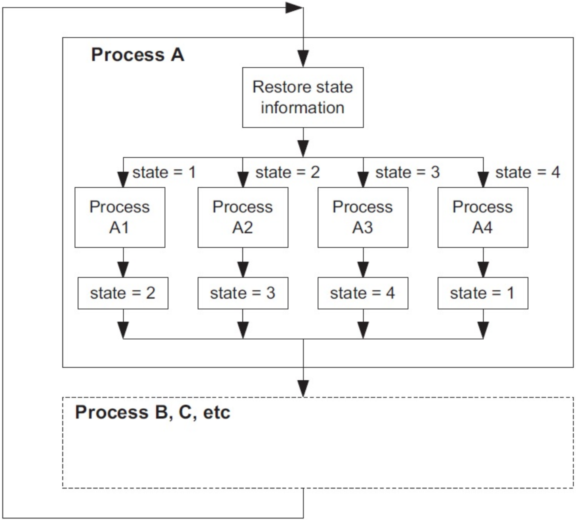


Figure: Interrupt driven Application

Concurrent Process handling



RTOS based Application

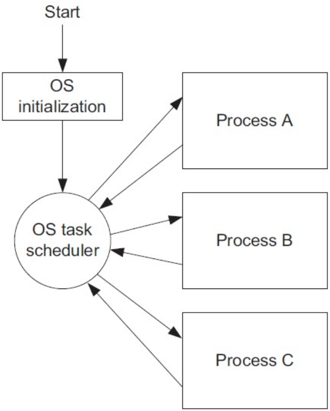


Figure: Real time Operating System

Programming Language for Embedded System Development

- Assembly language is a low-level programming language which is specific for a processor architecture.
- In contrast, high level language programs are easily portable across different architectures.
- Embedded processors e.g. 8051 have limited processing power and memory: the programs must be efficient.
- An assembly program provides complete and precise control of the available hardware resources.
- The hex file generated from an assembly language program will in general be smaller than the corresponding high level language.

C Data Types for the 8051

Data Types	Bits	Bytes	Value Range
bit †	1		0 to 1
signed char	8	1	-128 to +127
unsigned char	8	1	0 to 255
enum	8 / 16	1 or 2	-128 to +127 or –32768 to +32767
signed short	16	2	-32768 to +32767
unsigned short	16	2	0 to 65535
signed int	16	2	-32768 to +32767
unsigned int	16	2	0 to 65535
signed long	32	4	-2147483648 to +2147483647
unsigned long	32	4	0 to 4294967295
float	32	4	±1.175494E-38 to ±3.402823E+38
sbit †	1		0 or 1
sfr †	8	1	0 to 255
sfr16 †	16	2	0 to 65535

Figure: Data Types in C51 Compiler

Sbit Data Type

- Sbit is the widely used data type in C programs for accessing the single bit of the bit addressable special function register (SFR).

Write an 8051 C program to toggle bit D0 of the port P1 (P1.0) 50,000 times.

Solution:

```
#include <reg51.h>
sbit MYBIT = P1^0;    //notice that sbit is
                      //declared outside of main
void main(void)
  {
    unsigned int z;
    for (z=0; z<=50000; z++)
      {
        MYBIT = 0;
        MYBIT = 1;
      }
  }
```

Figure: Example of sbit data type

Bit Data Type

- You may use the bit data types for variable declarations, argument lists and function return values.
  E.g. bit mybit, static bit doneflag = 0

- All bit variables are stored in a bit segment in the 16 bytes of internal memory area of 8051. A maximum of 128 bit variables.

- Memory types of data or idata only may be included in the declaration.

- An array of type bit is invalid. A bit cannot be declared as a pointer.

- Functions that disable interrupts and functions that are declared using an explicit register bank can not return bit data type.

Special Function Registers

- SFRs are used in programs to control timers, counters, serial I/O, port I/O and other peripherals. Declarations for SFRs are provided in the include files for particular 8051 derivatives.
- SFRs reside from address 0x80 to 0xFF and can be accessed as bits, bytes and words.
- C51 compiler also provides access to SFRs with the sfr, sfr16 and sbit data types by using their direct addresses.
  - sfr P0 = 0x80; (Port 0 address 80h)
  - sfr16 T2 = 0xCC; (Timer 2, T2L address 0CCh and T2H 0CDh)
  - sbit EA = 0xAF; (SFR bit at address AFh)
  - sbit OV = 0XD0ˆ2; (PSW Registers 2nd bit)
  - sbit OV = 0XD2 ;(PSW Registers 2nd bit)

Memory Areas

- The 8051 architecture supports several physically separate memory areas for program and data.

Memory Type	Description
code	Program memory (64 KBytes); accessed by opcode MOVC @A+DPTR.
data	Directly addressable internal data memory; fastest access to variables (128 bytes).
idata	Indirectly addressable internal data memory; accessed across the full internal address space (256 bytes).
bdata	Bit-addressable internal data memory; supports mixed bit and byte access (16 bytes).
xdata	External data memory (64 KBytes); accessed by opcode MOVX @DPTR.
far	Extended RAM and ROM memory spaces (up to 16MB); accessed by user defined routines or specific chip extensions (Philips 80C51MX, Dallas 390).
pdata	Paged (256 bytes) external data memory; accessed by opcode MOVX @R*n*.

Figure: Explicitly declared memory types

Memory Areas

- You may specify where variables are stored by including a memory type specifier in the variable declaration. For example
  - char data var1;
  - char code text[ ] = "HELLO WORLD";
  - float idata x,y,z;
  - char bdata flags;
  - unsigned char xdata vector [10][4][2];

Memory Models

- The memory model determines the default memory type to use for function arguments, automatic variables and declarations with no explicit memory type specifier.
- In compact model, all variables by default reside in one page of external data memory. Maximum of 256 bytes of variable. Indirect addressing through R0 and R1.
- In Large model, all variables by default reside in external data memory. Maximum of 64 K bytes of variable. Indirect addressing through DPTR.
- In small model, all variables by default reside in the internal data memory. All objects as well as stack must fit into the internal RAM.

Bit Addressable Objects

- Bit-addressable objects are objects that can be addressed as words or as bits. Only data objects that occupy the bit-addressable area of the 8051 internal memory fall in this category.
- You may declare these variables as
  - int bdata ibase; (bit addressable integer)
  - char bdata bary[4]; (bit addressable array)
- You may use the sbit keyword to declare new variables that access the bits of variables declared using bdata. For example:
  - sbit mybit = ibase^7; (bit 15 of variable ibase))
  - ary07 = bary[0]^7; (bit 7 of bary[0])

Bit Addressable Objects

- Declarations involving the sbit type require that the base object be declared with the memory type bdata or it may be a special function register.
- You may declare these variables as external for the sbit type to access these types in other modules.
  - extern bit mybit0; (bit 0 of ibase)
  - extern bit ary07; (bit 7 of bary[0])
- You may not specify bit variables for the bit positions of a float.
- The sbit data type uses the specified variable as a base address and adds the bit position to obtain a physical address.

Pointers

- The C51 compiler supports the declaration of variable pointers using the $*$ character.
- C51 pointers can be used to perform all operations available in standard C.
- C51 compiler provides two different types of pointers: generic pointers and memory-specific pointers.
  - Generic pointer are similar to standard C pointers. e.g. char *s; (string pointer). These pointers may be used to access any variable regardless of its location in 8051 memory space.
  - Memory specific pointers always include a memory type specification in the pointer declaration and always refer to a specific memory area. e.g. char data *str; (pointer to string in data)

Logic operators in 8051 C

- The following bit-wise operations are used
- AND (&)
- OR (|)
- XOR ($\wedge$)
- Invert ($\sim$)
- Shift Right ($>>$)
- Shift left ($<<$)

Comments and Immediate data

- Keil accepts C or C++ style comments:

```
// this line will be ignored by the compiler

/* these lines will
   be ignored by the compiler */


unsigned char i;    // this is ignored
unsigned char j;    /* so is this */
```

- C format for decimal/hex/octal:

```
unsigned char i = 100;     // 100 as a base 10 literal
unsigned char j = 0x64;    // 100 in hex, indicated by leading 0x
unsigned char k = 0144;    // 100 in octal, indicated by the leading 0
```

Figure: Comments and literals

Creating Time Delays

- There are two ways of creating time delays: Software delay using simple for loop and 8051 timers
- Software delays using for loops depend on
  - Particular 8051 version
  - 8051 crystal frequency
  - Compiler choice
- The accurate number of iterations for loop for required amount of delay is computed using trial and error method.

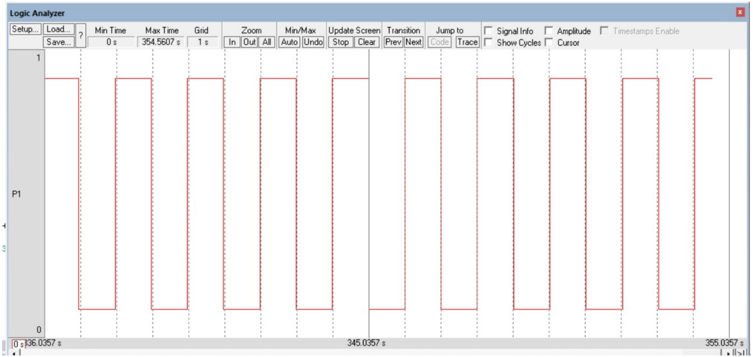
Creating Time Delays using for loop

```c
#include <reg51.h>
sbit portbit = P1^0;

void delay(unsigned int );

void main(void)
{
        while(1)
        {
                portbit = 1;
                delay(1000);
                portbit = 0;
                delay(1000);
        }
}

void delay(unsigned int MS)
{
 unsigned int x,y;
 for (x=0; x< MS; x++)
        {
                for (y=0; y<=113; y++);
        }
}
```

This function create a software delay of MS milisec, i.e. inner loop computes a delay of 1 milisec and the number 113 is computed using trial and error method.

Creating Time Delays using for loop

- We can see from the Keil logic analyzer that the delay is approx 1000 milisec.



Creating Time Delays using 8051 Timers

```
// This program toggles pin 1.1 continuously  every 250 ms //
// Using Timer 0, mode 2 (8 bit auto reload
// timer) XTAL = 12 MHz

#include <reg51.h>
sbit portbit = P1^1;
void timer0_delay( );
void main(void)
{
        unsigned char x,y;
        TMOD = 0X02;  // Setup timer 0 in mode 2
        TH0 = -25;    //Load THO with initial count
        while(1)
        {
                portbit = ~portbit;
                for (x=0; x<250; x++)
                  for (y=0; y<24; y++)
                    timer0_delay();
        }
}
```

```
// This function creates delay of 0.5 mili-sec using
// timer 0 in mode 1

void timer0_delay()
{
 TR0 = 1;     // Start the timer

        while (TF0!=1);  // wait for timer overflow

        TR0 = 0;    // Stop the timer
        TF0 = 0;    // Clear the timer overflow flag

 }
```

Serial Communication (Transmit)

```
// This program continuously transmit the message "HELLO"
// through serial  port at 9600 baud rate, 8 data bits, 1 stop bit
// i.e. (mode1) XTAL = 11.0592 MHz

#include <reg51.h>
void serial_tx(unsigned char);

void main(void)
{
        TMOD = 0X20;  // Set up timer 1 for baud rate
        TH1 = 0XFD;    // 9600 baud rate
        SCON = 0X50;  // mode1 i.e. 8 data bits, 1 stop bit
        TR1 = 1;       // start the timer

        while(1)
        {
                serial_tx('H'); // transmit one character
                serial_tx('E'); // We can also declare an
                                    array of char and use loop
                serial_tx('L');
                serial_tx('L');
                serial_tx('O');
        }
}
```

```
// This is the transmit subroutine for serial port
void serial_tx(unsigned char x)
{
        SBUF =x;      // write the character in SBUF
        while(TI!=1);  // wait for the transmission
                             to  complete
        TI=0;          // clear the transmit flag
}
```

Serial Communication (Receive)

```
// This program receives bytes of data through serial port and write these to
// port P1. Set the baud rate at 4800 bps, 8 bit data, 1 stop bit XTAL = 11.0592 MHz

#include <reg51.h>
void main{void}
{
        unsigned char rx_byte;
        TMOD = 0X20;   // Setup baud rate
        TH1 = 0XFA;
        SCON = 0X50;   // Serialcomm mode 1
        TR1 = 1;       // start timer

        while(1)
        {
                while(RI ~=1);  // wait for receive flag to be 1

          rx_byte = SBUF;  // Read the byte

                P1 = rx_byte;   // Write on P1 port

                RI =0;          // Clear the receive flag
        }
}
```

Interrupt Programming

- Interrupt service routine for each interrupt is extended by the keyword interrupt and an interrupt number.
- The interrupt number for each interrupt is given in the table below.

Interrupt number	Description
0	External0
1	Timer0
2	External1
3	Timer1
4	Serial port
5	Timer2

Interrupt Example

```
// This program uses timer0 and interrupt to create a
// 2 KHz square wave on pin P1.0, XTAL = 12 MHz

#include<reg51.h>
sbit portbit = P1^0;

void main()
{
        TMOD = 0X02;
        TH0 = -250;
        IE = 0X82;
        TR0 = 1;

        while(1);
}

void timer0_isr() interrupt 1
{
        portbit = ~portbit;
}
```

Interrupt Example

```
// This program continuously monitors pin P1.7 and sends it to P2.1. Simultaneously it
// creates a square wave of 200 micro sec period on pin P2.3 and sends letter "A" to
the // serial port. Use timer 0 in mode2 for delay and 9600 baud rate, mode one
// for serial communication. XTAL = 11.0592 MHz

#include<reg51.h>
sbit rx_bit = P1^7;
sbit tx_bit = P2^1;
sbit sq_wave = P2^3;
void main()
{
        rx_bit = 1; // Make this as an input
        TMOD = 0X22; // Setup timer
        TH0 = 0XA4;  // timer 0 in mode 2
        TH1 = -3;  // timer 1 for Baud rate setup
        IE = 0X92;  // Enable timer0/serial interrupt
        SCON = 0X50;  // Serial comm mode 1
        TR0 = 1;      //Start timer 0
        TH0 = 1;      // Start timer 1

        while(1)
        {
                rx_bit = tx_bit;  // Send the data
        }
}
```

```
void timer0_isr(void)  interrupt 1
        {
          sq_wave =~sq_wave;
        }

void seriial_isr(void) interrupt 4
    {
     if(TI ==1)  // if transmit interrupt
      {
       SBUF = 'A';  //Write  the byte
       TI = 0;     // clear the transmit flag
      }
     else
       {
       RI = 0;  // clear the receive flag
       }
    }
```