IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 5, Issue 6, Ver. II (Nov -Dec. 2015), PP 36-43
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Design of Real - Time Operating System Using Keil µVision Ide

Vijesh Mokati,Mtech(VLSI)
BMCT,Indore

Abstract: *Real time operating system (RTOS) is widely used in real time embedded system for computing resources management. It is commonly implemented as fundamental layer in software framework. Real-time operating systems must provide three specific functions with respect to tasks: scheduling, dispatching, and intercommunication and synchronization. In designing of RTOS, Here with Keil Ide, successfully implemented these basic needs. Its major point focuses on explaining the context switching by tricky usage of stack and synchronization mechanism.*

Keyword: *DCX, Mailbox, Nesting, Queue, Semaphore*.

I. Introduction

In some cases ,well structured linear programming is sufficient for programmers to appreciate not having to worry about structuring their code to perform all necessary tasks in a timely manner. This is where RTOS, can help.Here,I am presenting a Real Time Operating System for 8051 microcontroller series**, EMX.**
EMX is a tiny real time operating system based on priority, developed on the keil_v5.5 µvision .It is priority based ,having nested interrupts ,timer support and intercommunication with synchronization. Silent feature of **EMX** are :
1.A task is a never return function which loops forever.
2.Memory allocation at compile time.
3.Event management is neglected because of low RAM space availability of 8051.
4.Fast timing services
5.Compact and speedy Synchronization

II. Keil C51 Compiler's Setup

Because the limit size of 8051's RAM, Keil C extent the ANSI C to generate better codes which require less RAM and run faster. One extension is to let the compiler not to push all local register into memory stacks when a function is calling the other functions. This means that Keil C51 compiler allocates the local variables into fixed- addressing RAM. In another word, Keil C51 compiler let the local variables become global variable. This is a special case needed more attention. In the other side, it is easy to show that the RAM may have higher probability to run out if the program keeps on running different functions. Keil C51 compiler cures this by the way of **Data Overlaying Optimizing.** In Keil C51 compiler, for interrupt service routine, the contents of the R0 to R7, SFR ACC, B, DPH, DPL, and PSW, when required, are saved on the stack at function invocation time. This means when unnecessary, they will not be saved. So there is no way for **EMX** know how many and which registers are pushed into the stack. In order to make them all push the same registers when interrupts happen, and deal with the over-clever Keil C51 compiler's such unpredictable behaviors, it is necessary to add several lines of useless code in order to make the code more complex and force the Keil C51 Compiler push all registers for all ISRs. Such useless codes are put at the beginning of the,
void Int_Run(void) as shown In the table 1.1.Assembly coding done in C51,won't debug

```
void Int_Run(void )
{
//The following code is used for prevent the Keil C51's stupid optimization
INT8U* pt=0x06;
*pt = 0xff;
DPL =0x00;
DPH =0x00;
B =0x00;
PSW =0x00;
ACC =0x00
```

```
// The useful code start from here
}
```

in Evaluation software ,so care must be taken,SoPerformance Analysis of some different routine cannot be evaluated in demo version. Running with code size more than 2k is also not feasible with this evaluation software.

III. How To Use Emx

Like establish RTOS ,here also used of API, make the writing program code easy and for that different API have been discussed. Configuration can be user defined as well as functional ,where one can make decision according to choice.

A. EMX API

There are 16 API functions for EMX ,Which are tabulated below:

```
01   INT8U E_Version();
02   INT8U E_RunningTaskID();
03   void Int_Run (void);
04   void Int_Out(void);
05   void OSStart(void);
06   void OSInit(void);
07   void Del_Task(INT8U nTicks);
08   void MsgQ_Create(INT8U TaskID,INT8U* pMsgQ,INT8U Size);
09   void MsgQ_Post(UINT8* pMsgQ,INT8U Msg);
10   void MsgQ_Pend(INT8U* pMsgQ,INT8U* pRetV);
11   void Mailbox_Create(INT8U TaskID, Mailbox_t* pMailbox);
12   void Mailbox_Send(Mailbox_t* pMailbox,INT16U Msg);
13   void Mailbox_Receive(Mailbox_t* pMailbox,INT16U* pRetV);
14   void BinSem_Create(BinSem_t* pBinSem);
15   void BinSem_V(BinSem_t* pBinSem);
16   void BinSem_P(BinSem_t* pBinSem);
```

B. EMX Configuration:

```
// Functional Configuration                1 = Enable 0 = Disable
#define E_CONFIG_EN_MAILBOX                1  Support for mailbox ?
#define E_CONFIG_EN_MSGQ                   1 Support for message queue ?
#define E_CONFIG_EN_BIN_SEMAPHORE          1 Support for binary semaphore ?
#define E_CONFIG_EN_SYSTEM_TIMER           1  Support for system timing service?
#define E_CONFIG_EN_INT_NESTING            1  Support for interrupt nesting?
//Timer Setup
#define E_CONFIG_TICKS_CNT                 32 How many times of interrupt convert to one system ticks?
 //Task Information Setup
#define E_MAX_TASKS                        4  Number of Tasks
#define E_STAK_TASK_SIZE                   20  Stack size of Tasks (Unit: Byte)
```

C. User Configuration

These are the configuration to maintain at the run time of application. The application programmer could keep it intact or change it according to need of the program.

```
INT8U idata  Stack0    [E_TASK_STACK_SIZE];
INT8U idata  Stack1    [E_TASK_STACK_SIZE];
INT8U idata  Stack2    [E_TASK_STACK_SIZE];
INT8U idata  StackIdle[E_TASK_STACK_SIZE];
//Static Create Tasks ;As the need of program one can reduce the number of task
void Task0(void);
void Task1(void);
```

```
void Task2(void);
INT8U idata  Cod_task [E_MAX_TASKS]={Task0,Task1,Task2,E_IdleTask};
INT8U idata  Stak_task [E_MAX_TASKS]={Stack0,Stack1,Stack2,StackIdle};
// Maximum number of task can also be according to user choice, but maximum 4 no.
//Static Creation
D_Mlbox          Mailbox1;
D_BSem           BinSem1;
INT8U            MsgQ1[E_MSGQ_OFFSET+4];
```

IV. Designing With Keil

EMX contains priority based task scheduling, nesting interrupt supporting, timer functions as well as synchronizing mechanisms including critical section, binary semaphore, and mailbox and message queue. Please notice all the designs are served for the purpose of real application. Context switching is fast and routine has been in C programming . Event related routines and EDF scheduling cannot be implemented since **EMX** is primarily designed for 8051 and 89S52 which having small RAM size .

A. Task & Static Memory Allocation

As in other RTOS, a task in EMX is simply a never return function which loops forever. Besides, every task has its own stack which is defined by the application programmer. Memory allocations of EMX are done in compile time. The task stack is defined explicitly as an array. As there is only very little RAM space for data, cannot possible to allocate too much for tasks stack. But there is a lower bound of the stack size because interrupt may happen during the executing time of the task and nested interrupt may even happened. ISRs may be called many times and they will push registers into the stack of the task that is running. For 8051, there are at most two levels of nesting interrupts may happen. Each ISR will push 13 Bytes registers and 2 Bytes of Return address into the stack. A third level of nesting is also possible but here we use only two level has been employed, So in case of nesting interrupt, at least allocate (13+2)*2 = 30 Bytes for the stacks of tasks. It is the application programmers' job to put the address of the entry code for each tasks into the Cod_task array. Also, as in many RTOS, EMX does not support tasks killing and assumes that each task will loop forever.

B Context Switching & Stacks

When context switching happens, it is necessary to push all register into the task stack. Also the stack pointer will be saved in EMX which will retrieve back to resume the right information. The code uses **src** directive of Keil µvision and produces C code for assembly routines. Here two API's namely, Save_Context() and Load_Context() used in conjunction . In Save_Context () ,running task PSW,ACC,B,DPL, DPH,SP and registers are pushes on the stack, while in Load_Context all of these registers

Pop back to continue the task completion. Another important thing is that all the above procedures must be guarded by macros E_ENTER_CRITICAL() & E_EXIT_CRITICAL() of the since any interrupt interference will cost disaster error here. These macros can be defined as

```
#define  E_ENTER_CRITICAL()          Disable_Interrupt(), Critical_Sum++
#define  E_EXIT_CRITICAL()           if (Critical_Sum==0) Enable_Interrupt()
```

we need the variable Critical_Sum to record the levels of Critical Section the CPU enter
This is used to solve the potential problem of stack overflow. Important point is here both of these macros must be used in tandem.

C. Priority Based Scheduling & Distributed TCB

In fact, the scheduler of tasks is very simple. It just need to check bit by bit whether a task is in ready state from higher priority to lower priority ones. The real codes are not shown here, but in example program mention how to use the scheduler. In the implementation of EMX there is no explicit Task Control Block in the data structure of RTOS. In EMX , we store them like this:

1. Task PC Value In the stack (PUSH inexplicitly by hardware when MCU is executing CALL instruction or hardware interrupt happens)
2. Task Stack Pointer Value In the array Stak_Task[]
3. Task Registers Value In the stack (PUSH explicitly by assembly language)
4. Task State In the word E_TaskCond .

C. Interrupt & Context Switching

The interrupt invoking attribute uses keil C51interrupt calling procedure embracing ISR with RTOS Define API. The codes of interrupt invoking are tabulated below:

```
#pragma disable
void userISR(void) interrupt N {
    Int_Run() ;
  //user ISR code here
     Int_Out ();   }
void Int_Run (void); {
E_IntNesting++;
Save_Context_Int()
Enable_Interrupt();   }
void Int_Out(void   {
E_ENTER_CRITICAL();
E_IntNesting--;
if(E_IntNesting==0)  {
E_Scheduler();
Load_Context_Int(); // Gone and never return
}else
{
   Critical_Sum--; // =  OS_EXIT_CRITICAL() + Disable_Interrupt()
}
}
```

Here Save_Context_Int () do not need to push the register on to the stack and Load_Context_Int () need to run the instruction RETI .

D. Interrupt Nesting

Nested interrupts is supported by recording the nesting count in E_IntNesting. Then when the ISR call Int_Out () before return, the RTOS will make use of E_Int Nesting to determine whether performing scheduling algorithm, load new context and switch to the new task or just simply come back to the low priority level ISR.
A tricky technique is used to reduce the Critical count Critical_Sum while still keep on disabling interrupt before POP back all registers. The Keil C51 compiler then will generate the code to pop back all register and after that a "SETB EA" to enable the interrupt again. If the code enable interrupt by calling OS_EXIT_CRITICAL() , the interrupt is enable when the CPU is popping back the registers.

E. Timing Service

EMX provides the timing service by the API "void Del_Task(INT8U nTicks)" which will delay the called task but nTicks numbers of Ticks. The unit of nTicks, Tick, include. E_CONFIG_TICKS_CNT numbers of happening of E_Timer0_ISR(). This mechanism is to prevent too frequent calling the E_TimeWake () functions which may waste lots of time, i.e., to slow down the timer in case you do not need it to run so quickly.

```
void Del_Task(INT8U nTicks)
{
if(nTicks==0) // prevent dead lock happen
return;
OS_ENTER_CRITICAL();
RestTick[PresentTask] = nTicks;
TaskCond = TaskCond & (~(0x01<<PresentTask));
Save_Context();
E_Scheduler();
Load_Context(); //Never return
}
```

E_TimeWake(void) is used to awake the sleeping blocked tasks if the time for them to wake up arrives. It reduces the count in the RestTick [ ] array and check whether it is equal to zero. If yes, it will change the state of

the that task into ready state.API function "void Del_Task(INT8U nTicks)" is used to set the remaining time in the RestTick[ ] array and change the running task into ready state.

F. Binary Semaphore

Binary semaphore is the most simple synchronization mechanism in **EMX**. At the same time, it is also the most efficient one in **EMX**. It will cost only 1 Byte RAM.
The usage example:

```
D_BSem    BinSem1;
BinSem_Create(&BinSem1);
BinSem_V(&BinSem1);
BinSem_P(&BinSem1);
```

One bit of the semaphore is used for indicating whether the corresponding task is waiting or not. If yes, the corresponding bit will be set to 1. Since the Idle Task will not try to get the semaphore, so the most significant bit which is corresponding to the idle task is used to indicate the value of the semaphore.
The Keil C51 µvision program for Binary Semaphore creation is shown below and

```
Typedef    D_BSem      pBinSem
#if OS_CONFIG_EN_BIN_SEMAPHORE > 0u
D_BSem     BinSem_Create( INT8U* pBinSem) { *pBinSem = 0x80;}
#endif
```

Performance Analysis figure also shown in the respective figure 1. The such analysis for all the API related to Binary Semaphore can be done and find the response of them.

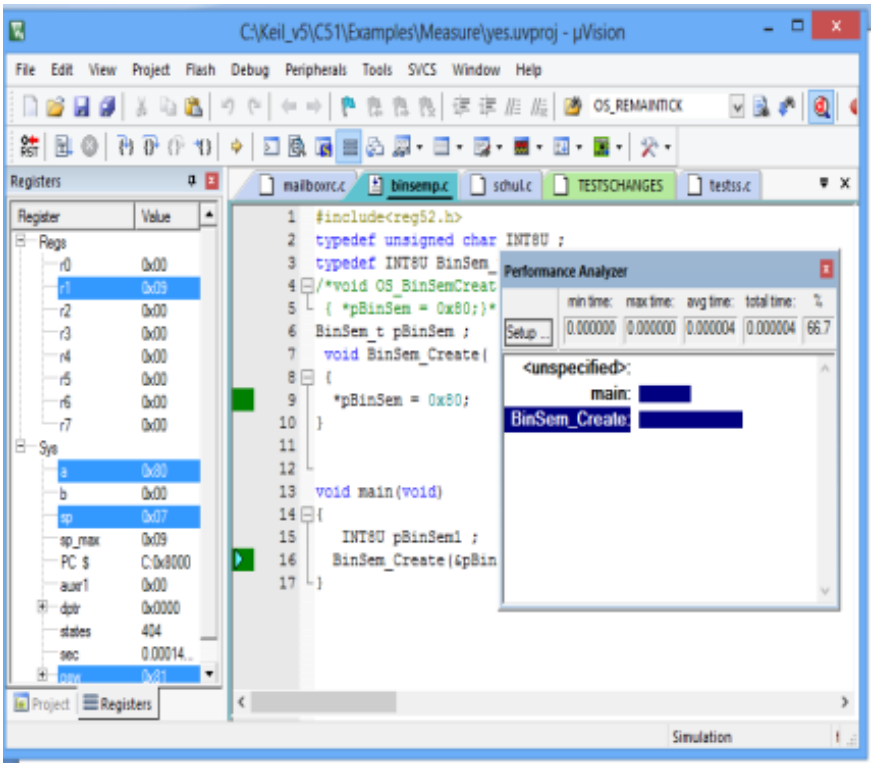

Figure.1 Performance Analysis of BinSemCreate.

G. Mailbox

A mailbox needs 4 Bytes to store the information which contains 2 Bytes for message content, 1 Byte for the pointer of return address, and 1 Byte for saving the task ID of the task that is waiting.
The usage of Mailbox can be accomplished as shown below:

```
D_Mlbox     Mailbox1;
```

```
Mailbox_Create(0,&Mailbox1);
Mailbox_Send (&Mailbox1,0x1234);
Mailbox_Receive(&Mailbox1,&rcvMsg);
```

Here ,presented the Performance analysis of Mailbox_Receive () with µvision and the respective figure is shown in figure 2.Similar Performance analysis of different API related to Mailbox operation can also be possible. The special care must be taken for assembly routine written in C51 compiler.

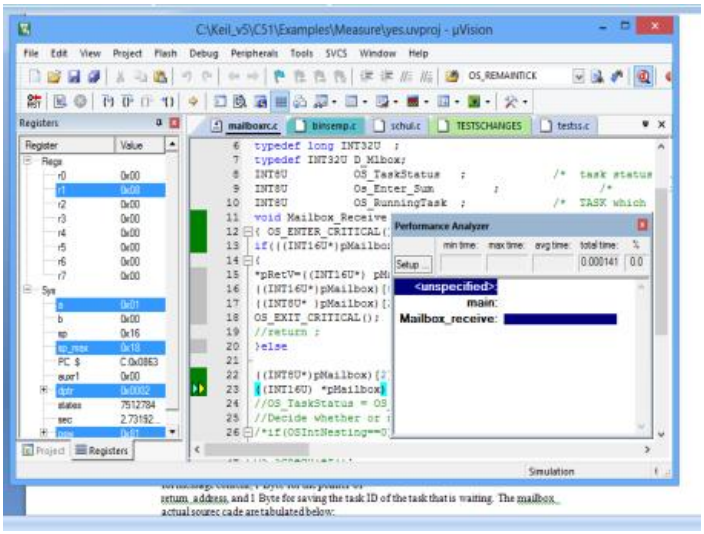

Figure .2 Performance analysis for API Mailbox_Receive()

H. Message Queue

The message queue is kind of RAM consuming mechanism. At least 6 Bytes are needed to implement a message queue. The message size for message queue is 1 Bytes which is a half of the message size in mailbox. The API related to Message Queue are tabulated below with performance analysis of one of them and the figure.3 give the detail of API void MsgQ_Pend().

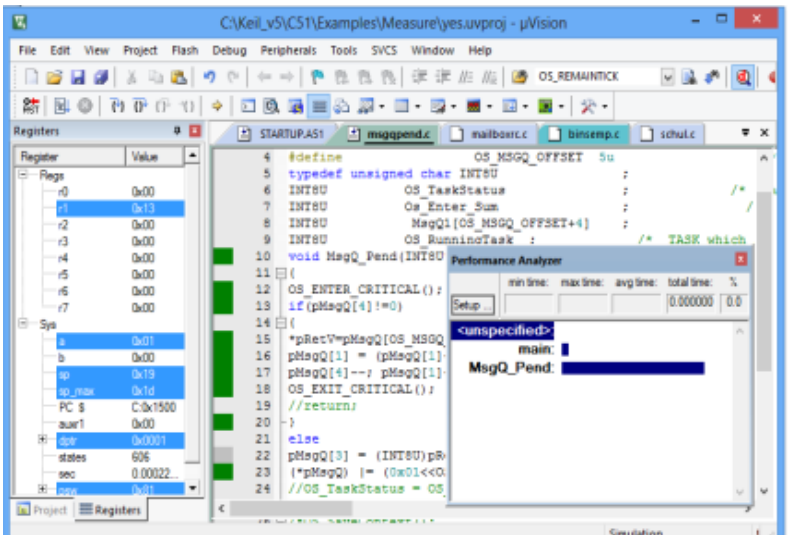

Figure .3: Performance Analysis of API MsgQ_Pend().

H. Idle Tasks

This is the simplest part of **EMX**. The code for Idle Task is tabulated here for reference .

```
void E_IdleTask(void) { while(1){ PCON ! =0x01; /* Save Power and Statistic*/; }
```

V. Example

Here ,an example presented which provide an insight to use of **EMX** RTOS. In this example ,created the Semaphore,Mailbox,and Message Queue ,initialize one of the timer of 8051 and created a task .In this task ,use of reception of a message and then forward it to Port 0. The full Program is given below for the reference and its debug analysis is also presented .The Code coverage facility provided by µvision Provides statistical data about the execution of the application. Execution analysis reports can be viewed and printed for certification requirements.

```
 #pragma disable
    void myISR1(void) interrupt 0
{     void Int_Run (void);
    Mailbox_Send(&Mailbox1,0x2234);
   void Int_Out(void);}
void Task0(void)
{       IN16U  rcvMsg;
   while(1){
   Mailbox_Receive(&Mailbox1,&rcvMsg);
   P1=rcvMsg/256;
    P0=rcvMsg%256;
   Del_Task (1);}}
void main(void)
{ OSInit();
Mailbox_Create(0,&Mailbox1);
BinSem_Create(&BinSem1);
MsgQ_Create(2,MsgQ1,4);
PX0=0; IT0=1; PX1=1; IT1=1;                          //Extern Interrupt
TMOD = (TMOD & 0XF0) | 0X01;                         //System Timer Setup
TH0 = 0x0; TL0 = 0x0;                                //Time interval = 0xFFFF
TR0 = 1;
ET0 = 1;
PT0 = 1;
OSStart();
}
```

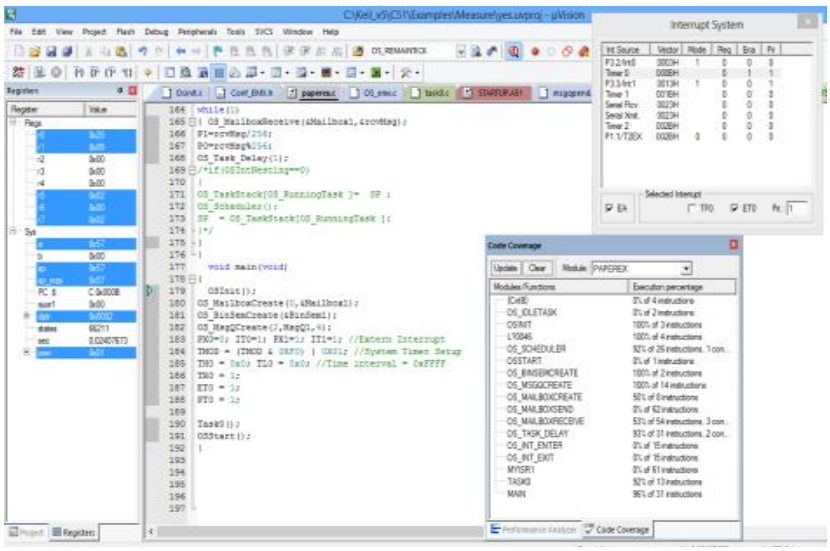

Figure. 4: Code coverage of Example

VI. Conclusion

Keil C51 complier together with uVision IDE, the most popular developing environment around the world for 8051, offers an excellent programming & debugging environment, as well as very precise simulator. The EMX simulate here pretty well and some examples of synchronization and inter task communication have been taken .Their response is quite encouraging. The version of my complier and simulator are Keil C51 V9.53.0.0 and µVision V5.11.2.0. The target MCU is the most popular 8051 series MCU -- Atmel AT89S52.

References

[1]. Schulz,C and the 8051
[2]. Michael J. Pont,Embedded C
[3]. Jean J Labrosse, MicroC/OS, The Real-Time Kernel
[4]. Ganssle,Jack G.,The Art of programming Embedded System
[5]. McConnell,Steve Code-Complete, A Practical Handbook of Software Construction
[6]. Jean J.Labrosse, Embedded System Building Blocks,
[7]. Gerald Jochum,Emmanuel Tuazon and Subramaniam ,Ganesan Traffic Light Implementation Using uC/OS
[8]. Richard A. Burgess,MMURTL V1.0,
[9]. ARM-uvision_Tutorial
[10]. Phillip A. Laplante Real -Time System Design and Analysis
[11]. David Kallinsky ,Basic Concept of real-time operating systems,Windows Embedded
[12]. Micrium Corporation uC/OS-III,The Real-Time Kernel,users manual
[13]. Qing Li and Carolyn Yao Real –Time Concepts for Embedded System
[14]. Wikibooks ,Embedded System,Building and Programming Embedded Devices, 15. Giorgio C. Buttazzo ,Hard Real-Time Computing System : Predictable Scheduling algorithms and Application
[15]. Donal Huffernan 8051 Tutorial,
[16]. Wind River VxWorks 6.0 Harware Consideration Guide
[17]. 80C51 Family Hardware description,Philips Semiconductor
[18]. Jane_W_S_Liu,Real-Time System
[19]. Rajib Mall , Real-Time Systems and Theory
[20]. Ke Yu, Real-Time Operating System Modeling and Simulation,using SystemC
[21]. XIAO, Jianxiong Project Report of a Small Real Time Operating System – USTOS
[22]. Raj Kamal, . Embedded Systems: Architecture, Programming and Design
[23]. Jogesh K. Muppala, HKUST COMP355 Embedded System Software Lecture Notes andCourse Materials
[24]. Keil Software Corporation: User's Guide of Cx51 Compiler, http://www.keil.com/
[25]. Atmel Corporation: Manuals of AT89S52 chip
[26]. David E. Simon, An Embedded Software Primer
[27]. Muhammad Ali Mazidi,Janice Gillispie Mazidi Rolin D. McKinlay, The 8051 Microcontroller and Embedded Systems Using Assembly and C