RESEARCH & PROJECT SUBMISSIONS

Program:

Course Code: CSE312

Course Name: Microprocessor Based Systems

Examination Committee

Dr. Ashraf M. M. El farghly Salem.

Ain Shams University
Faculty of Engineering
Spring Semester – 2020

Student Personal Information

Student Name: محمد عمرو احمد ظة منسي
Student Code: 1601247
Class/Year: 3rd electrical year computer department section:3

Plagiarism Statement

I certify that this assignment / report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment / report has not been previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons.

Signature/Student Name: محمد عمرو احمد طة منسي **Date: 7/6/2020**

Submission Contents

01: Comparison of three RTOS

02: project code

Comparison between three RTOS

Free-Rtos:

1-Multithreading and Scheduling:

-Prioritized Pre-emptive Scheduling with Time Slicing
-> *Fixed Priority* the tasks being scheduled the priority assigned to it doesn't change thus we can change task own priority
-> *Pre-emptive* scheduling is preempt the task in running state if task is come in ready state with higher priority
-> *Time slicing*: is used to share the *task with* same priority processing time in cpu even the tasks do not yield or arrive the Blocked state.

-Prioritized Pre-emptive Scheduling without Time slice. It same as algorithm descript in above but without slicing. is used to share the *task with* same priority processing time in cpu
- co-operative scheduler:at this scheduler the context switch will only occur when The task in running state enter the blocked state or the task in running state is yield by taskYIELD(). The time slicing is used due to tasks pre-empted,

2-Tick:

- The RTOS tick may be a settled interim intermittent interrupt produced shape a equipment timer. It is utilized to degree genuine time. The determination of time that can be measured is subordinate on the tick period.
- For illustration, in the event that the RTOS tick recurrence is set to 1ms at that point time can be measured with an exactness of 1ms but there's too an overhead of having an interrupt each 1ms. If this determination isn't essential (it ordinarily isnt) at that point the overhead can be decreased by abating down the tick recurrence to say 10 or indeed 100ms.
- The tick interrupt is additionally utilized for round robin planning. When there's more than one errand that have the same need and are prepared to run, at that point the tick hinder is utilized to switch between the assignment

3- Task Management:

- Tasks are implemented as C functions which must return void and Take a void pointer parameter.
- Tasks are created by using the FreeRTOS xTaskCreate() API function. Creatingtask:
- the scheduler choose the task in ready state that have highest priority and enter it to the running state.
- task can delay in blocked state wait for event and when event happen it's are automatically return back to the Ready state.

4-Hardware Abstraction layer:

Hal is:main interface for all drivers, there are two function (init-deinit) and hal struct is should be implemented in all driver interfaces.1-init :return a handler access instances of the driver 2-deinit:disable that instances of driver , to create a new entry we use HAL_ADD()and the linker script must have Dev_DEFAULT()

- ❖ Notice that: All instants of driver are statically allocated. – must include hal .h

5-ApiandModules:

Queues module: Detailed Description for five API all of them is in queue.h

1-uxQueueMessagesWaiting

-UBaseType_t uxQueueMessagesWaiting(QueueHandle_t xQueue);

-it'sReturn the number of messages which stored in a queue.

-Parameters: xQueue A handle to the queue that be queried

2- uxQueueSpacesAvailable:

-UBaseType_t uxQueueSpacesAvailable(QueueHandle_t xQueue);

-Return the number of free spaces that is in a queue.

-Parameters: XQueue handle to the queue that be queried

3-vQueueDelete:

-void vQueueDelete(QueueHandle_t xQueue);

-Delete a queue in which freeing all the memory allocated for storing of items that is place on the queue.

-Parameters: Xqueue is A handle to the queue to can be deleted

4- xQueueReset:

-BaseType_t xQueueReset(QueueHandle_t xQueue);

-Resets a queue to its original empty state.

-Parameters:XQueue the handel of the queue that be rest

Returns: note that FreeRTOS V7.2.0 xQueueReset() continuously returns pdPASS

5- xQueueIsQueueEmptyFromISR:

-BaseType_t xQueueIsQueueEmptyFromISR(const QueueHandle_t xQueue);

-Enquiries a queue to determine if the queue is empty. This function should only used in an ISR.

-Parameters:xqueue the handel of queue that be queried

-Returns: pdFALSE when the queue is not empty, or pdTRUE when the queue is empty

6-RTOS Configuration:

- FreeRTOS is redone employing a Configuration file called FreeRTOSConfig.h. Each FreeRTOS application must have a FreeRTOSConfig.h header file in its pre-processor incorporate way. FreeRTOSConfig.h tailors the RTOS part to the application being built. It is subsequently particular to the application, not the RTOS, and ought to be found in an application registry, not in one of the RTOS kernel source code catalogs.
- Each demo application included within the RTOS source code download has its possess FreeRTOSConfig.h record. A few of the demos are quite ancient and don't contain all the accessible arrangement alternatives. Arrangement choices that are overlooked are set to a default value inside an RTOS source file.
- Here is a typical FreeRTOSConfig.h definition, followed by an explanation of each parameter: #ifndef FREERTOS_CONFIG_H, #define FREERTOS_CONFIG_H.

Ti-Rtos:

1-Multithreading and Scheduling:

- TI-RTOS Kernel gives back for a few distinctive sorts of threads in an embedded system.
- Hardware Interrupt (Hwi): back threads started by a hardware interrupt.

- Software Interrupt (Swi): organized to be comparative to Hwis, but permit handling to be conceded until after a hardware interrupt has completed.
- Task: a discrete thread that can execute or block whereas holding up for an occasion to happen.
- -Idle: the least need thread that as it were runs when no other thread is prepared to execute.

2-Tick:

Overview of time services:

Several modules are involved in timekeeping and clock-related to SYS/BIOS and XDCtools services

-ti.sysbios.knl.Clock module:which is responsible for the periodic system tick the kernel use it to keep track of time. and used to schedule functions that is run at intervals in clock ticks and uses the Hal. Timer module to get a hardware-based tick.
-The ti.sysbios.hal.Timer module: providing a standard interface to use it for timer peripherals.
-The ti.sysbios.hal.Seconds module **:**providing a means for maintaining the current time and date,
defining by number of seconds.
-The xdc.runtime.Timestamp module **:**providing simple timestamping services for benchmarking code and timestampsis adding to logs.

3- Task Management:

Execution of States and it's Scheduling: there are many modes is (RUNNING-READY-BLOCKED**-**TERMINATED-INACTIVE**).**

1-Running: refer to that the task is executing in cpu(processor) .

2-Ready: refer to that the task is ready and wait to be scheduled in the cpu when it's available.

3-Blocked: refer to that the task is waiting for event or i/o resources and doesn't execute until event happen.

4-Terminated: refer to that the task is finish and doesn't run anymore.

- Tasks are scheduled according to priority level add by application, may be no more than one

Runningtask ,no ready task have priority level high than current running task due to Task preempts the running task on the side of the higher-priority ready task, unlike many system that give

Fair time share in cpu for al task, when a task of higher priority becomes ready to Run,the SYS/BIOS *instantly* preempts the current task

- First the task created by Task_create() then enter ready mode wait then enter run mode and can be preempted by high priority ready task and return to ready mode by Task_yield() and then after time slice end go to terminated mode or is suspends due to need i/o resources Or sleep so go to blocked

mode by Task_sleep() or Semaphore_pend(), when task is ready go to run mode from blocked by Semaphore_post(), can task be deleted and go to terminated mode by Task_delete().

4-Hardware Abstraction layer:

- SYS/BIOS gives services for arrangement and services of interrupts, cache, and timers Unlike other SYS/BIOS services such as threading, these modules specifically program angles of a device's hardware and are gathered together within the Equipment Reflection Layer (HAL) bundle. services such as enabling and debilitating hinders, stopping of interrupts vectors, multiplexing of numerous interrupts to a single vector, and cache nullification or writeback

➢ Generic APIs that are accessible over all targets and devices:

Which: Developers are concerned with guaranteeing simple portability between diverse TI devices are best served by

utilizing the bland APIs as much as conceivable.

➢ Target/device-specific APIs that are accessible as it were for a particular device or ISA family:

Which: give full hardware entitlement.

When it's used?! Used when a device-specific hardware feature is found which is advantageous to software application,

5-APIs and Module :

-***Camera:*** ***Camera_init()** initialize the Camera module,***Camera_Params_init()** initializes an Camera_Params data structure.
-GPIO: **GPIO_init()** configure GPIO pins,* **GPIO_read()** gets the current state of GPIO input pin.

6- RTOS configuration:

-can configuring SYS/BIOS applications by modifying the *.cfg configuration file in the project

- in which this file is written in the XDCscript language, in which CCS provides a Graphical configuration editor called XGCONF,in which can edit the file by using the XGCONF GUI interface. that is XGCONF is the default editor for files listed in the Project Explorer window, we can choose the required components simply by clicking the icon, or configure the parameters by choosing the related options
-The aim of making SYS/BIOS:
1-determint the used modules and packages by application.
2-making objects for the modules that used by application.
3-for changing their runtime behavior it by setting parameters for the system, modules, and objects

Keil-Rtx

: 1-Multithreading and Scheduling:

-**Pre-emptive scheduling**:

1.1- RTX could be a pre-emptive multitasking operating framework. In case a task with the next need than the right now running
task gets to be prepared to run, RTX suspends the as of now running task

2.1- Each task contains a distinctive need and will run until it is pre-empted or has come to a blocking OS call.

.- Round-Robin scheduling:

1.1- Each task encompasses a distinctive need and will run until it is pre-empted or has come to a blocking OS call

.1.2--RTX can be arranged to utilize Round-Robin Multitasking (or task exchanging).

-**Round-Robin** permits quasi-parallel execution of a few tasks. Tasks are not truly executed concurrently but are time-sliced (the accessible CPU time is separated into time cuts and RTX allocates a time cut to each task). Since the time cut is brief (as it were a couple of milliseconds) it shows up as in spite of the fact that task execute at the same time. Notice that the default algorithm is Round-Robin

-**Co-operative multi-tasking**:

1.1- Each task has the same need and the Round-Robin is debilitated. Each assignment will run until it come to a blocking OS call or employments the os_tsk_pass() call.

1.2- On the off chance that you impair Round-Robin Multitasking you must plan and actualize your tasks so that they function agreeably. Particularly, you must call the framework hold up function just like the os_dly_wait()function or the os_tsk_pass()function some place in each task These capacities flag the RTX bit to switch to another task.

```
#include <rtl.h>
OS_TID id1, id2;
void task1 (void){
id1 = os_tsk_self();
id2 = os_tsk_create (task2, 0);
for (;;) {
    os_evt_set(0x0004, id2);//Signal to task2 that task1 has complete
 // place code for task1 activity here
 os_evt_wait_or(0x0004, 0xFFFF); //Wait for completion of task1 activity
os_dly_wait(5);
  }}
void task2 (void) {
  for (;;) {
os_evt_wait_or(0x0004, 0xFFFF); //Wait for completion of task2 activity
os_dly_wait(2);
 // place code for task2 activity here.
 os_evt_set(0x0004, id1); // Signal to task1 that task2 has completed
  }}
 void main (void) { os sys init(task1);}
```

```
// here code of timer interrupt
Handler in assembly in keil

void PendSV_Handler (void){
EL cpp(rt_pop_req)
LDR R3,=cpp(&os_tsk)
LDM R3, {R1, R2}
CMP R1, R2
BEQ Sys_Exit
PUSH  {R2,R3}
MOV   R3, #0
STRB  R3, [R1, #TCB_RETUPD]
MRS   R12, PSP
```

```
STMDB R12!, {R4-R11}
STR   R12, [R1, #TCB_TSTACK]
BL    rt_stk_check
POP   {R2, R3}
STR R2, [R3]
LDR R12, [R2, #TCB_TSTACK]
LDMIA R12!, {R4-R11}
MSR PSP, R12
LDRB R3, [R2, #TCB_RETUPD]
CBZ R3, Sys_Exit
LDRB R3, {R2, #TCB_RETVAL}
STR R3, [R12]
Sys_Exit MVN LR, #:NOT:0xFFFFFFFD
BX LR}
```

2-Tick:

Timer Tick Interrupt :

- The RTX bit for ARM7™ and ARM9™ employments one of the standard ARM timer to produce a occasional hinder. RTX kernel for Cortex™-M employments common SysTick timer. This intrupte is called the RTX kernel timer tick. For a few of the RTX library capacities, you must indicate timeouts and delay interims in number of RTX kernel timer ticks.

Alternate Tick Timer:

- RTX Library version for Cortex™-M devices employments SysTick timer as RTX tick clock. The SysTick clock is Cortex-M center timer in this way common for all Cortex-M device variations. A few unused double center devices, such as LPC4300 devices, don't execute the SysTick timer in both centers. In this case, an interchange tick clock must be utilized for the center without SysTick timer.

Interface:

os_tick_int()-os_tick_irqack()-os_tick_handler().

3- Task Management:

- Each RTX task is continuously in precisely one state, which tells the mien of the task

So we have:

- **RUNNING:** The task that's currently running is within the RUNNING state. As it were one task at a time can be in this state. The os_tsk_self() returns the Task ID (TID) of the as of now executing task. .
- **READY** :Tasks which are prepared to run are within the Prepared state. Once the running task has completed preparing, RTX chooses the following prepared task with the most noteworthy need and begins it..
- **WAIT_DLY:** Tasks which are holding up for a delay to run out are within the WAIT_DLY State. Once the delay has terminated, the task is exchanged to the Prepared state. The os_dly_wait() work is utilized to put a task within the WAIT_DLY state.
- **WAIT_ITV** Tasks which are holding up for an interim to run out are within the WAIT_ITV State. Once the interim delay has terminated, the task is exchanged back to the Prepared State. The os_itv_wait() work is utilized to put a task within the WAIT_IVL State.

4-Hardware Abstraction layer:

Hal is means : interface for all driver in which all APIs that STMicroelectronics have for microcontrollers is either called Hardware Abstraction layer to initializing their peripherals must configure CMSIS v2.0 compliant programs

5-APIs and Module :

1:-Kernel Information and Control: *osStatus_t osKernelInitialize (void) Set the RTOS Kernel *osKernelState_t osKernelGetState (void), Get the recent RTOS Kernel state.

2:- Thread Management: *const char * osThreadGetName (osThreadId_t thread_id) Get name of a thread.*osThreadId_t osThreadGetId (void) Return the thread ID of the current running thread.

3:-Timer Management : 1- const char * osTimerGetName (osTimerId_t timer_id)Get name of a time.*osStatus_t osTimerStop (osTimerId_t timer_id) Stop a timer

6- RTOS configuration:

Tasks:

- OS_TASKCNT indicates the greatest number of task that can be dynamic at the same time. This incorporates task in any state (running, holding up, or prepared) other than the INACTIVE state.
- This data is utilized by the RTX kernel to save the memory pool for the task control factors. This number can be higher than the number of characterized task in your application (one task can run in numerous occurrences) or lower in the event that it is ensured that the number of made and running tasks will never surpass OS_TASKCNT.
- #define OS_TASKCNT 6

➢ **OS_PRIVCNT** specifies the number of tasks with **user-provided** stack.

- The term user-provided, in this case, implies that the memory space for the task's stack is given by the client when the task is made. It isn't naturally allocted by the kernel. The RTX bit employments OS_PRIVCNT to optimize the memory utilization. The bit will not save stack space for the tasks with a user-provided stack.
- #define OS__PRIVCNT 0

Stack Size:

- The stack utilization of a specific task depends on its sum of neighborhood programmed factors and the number of subroutine levels. On task switch, the RTX bit stores moreover ARM registers on the stack. Devices with Equipment Drifting Point unit require extra space for putting away the VFP registers. The estimate of stack utilized for the setting spare is:

➢ OS_STKSIZE indicates the sum of Smash apportioned for the stack of each task. Stack measure is characterized in U32 (unsigned int). Be that as it may, Arrangement Wizard changes over the desired measure and shows it in bytes. Stack measure with the taking after characterize is 400 bytes.

➢ `#define OS_STKSIZE 100`

Stack Checking

- On the off chance that Stack Checking is empowered, the kernel can identify the stack depletion issue and execute the framework mistake function. The application will hang in an unending circle interior the mistake function with parameter err_code set to OS_ERR_STK_OVF. You'll distinguish the task id with isr_tsk_get() function Check the Dynamic task investigate discourse for the task title.

- The arrangement to this issue is to extend the stack size for all tasks within the setup file. On the off chance that as it were one task requires a huge stack and ram is restricted, you'll make this task with a user-provided stack space.
- **OS_STKCHECK** enables the Stack Checking algorithm. It must be set to 1 to empower it or to debilitate it. It is enabled by default.
- `#define OS_STKCHECK 1`

Run in Privileged Mode

- RTX Library adaptation for Cortex™-M devices permits to choose the running mode of all client tasks. Client tasks may run in two modes.

➢ **Unprivileged** _ Protected mode or
➢ **Privileged** _Unprotected mode.

- in favored mode client may get to and design the framework(system) and control registers like NVIC intrupte controller etc. Usually in any case not permitted from unprivileged mode. An get to NVIC registers from unprivileged mode will result in Difficult Hard Fault.
- OS_RUNPRIV empowers running of all tasks in Advantaged mode. It must be set to 1 to empower it or to debilitate it. It is impaired by default
- `#define OS_RUNPRIV 1`

Hardware Timer

The following options can be configured:

- OS_TIMER indicates the on-chip timer utilized as a time-base for the real-time framework. It conveys a intermittent interrupt that

wakes up a time-keeping framework task The client can select which timer serves this reason:

➢ ARM7/ARM9 library version: Utilize for Timer 0, or 1 for Timer 1, etc
.
➢ **Cortex-M** library version: Utilize for Center SysTick Timer, or 1 for substitute Fringe Timer.

```
#define OS_TIMER  1
```

➢ **OS_CLOCK** indicates the input clock recurrence for the chosen timer.
```
#define OS_CLOCK
15000000
```
➢ **OS_TICK:** shows the timer tick intervals in µsec. Endorsed values are 1000 to 100000. The coming approximately intervals is from 1 ms to 100 ms. Default course of action is for 10 ms.
➢ `#define OS_TICK 10000`

Round-Robin Multitasking

the following options can be configured:

➢ **OS_ROBIN** empowers the Round-Robin Multitasking. It must be set to 1 to empower it or 0 to disable it. It is empowered by default.
➢ `#define OS_ROBIN 1`
➢ **OS_ROBINTOUT** indicates the Round-Robin Timeout. Typically the time-slice doled out to the right now running task. After this time-slice terminates, the as of now running task is suspended and the another task prepared to run is continued. It is indicated in number of framework clock ticks.
➢ `#define OS_ROBINTOUT 5`

User Timers

- **OS_TIMERCNT** indicates the number of client timer that can be made and begun. In case client clocks are not utilized, set this value to 0. This data is utilized by RTX to save the memory resources for timer control block
- `#define OS_TIMERCNT 5`
- The callback function os_tmr_call() is called when the client timer terminates. It is given within the RTX_Config.c configuration file as an purge function You must modify it to suit your needs.

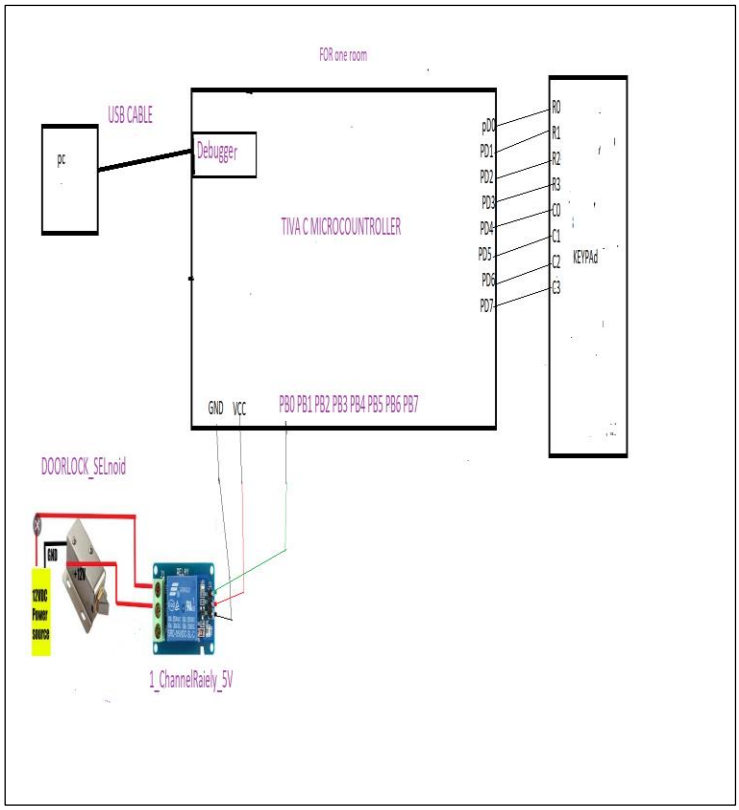
- Parameter information is the parameter passed to the os_tmr_create() function when the timer was made.

FIFO Queue Buffer

- the isr_ library function, when called from the interrupt handler, stores the ask type and discretionary parameter to the ISR FIFO Queue buffer to be handled afterward, after the interrupt handler exits.
- The task manger is enacted instantly after the IRQ handler has wrapped up its execution to handle the requests put away to the FIFO Queue buffer. The measure of this buffer required, depends on the number of isr_ capacities, that are called inside the interrupt handler
- For illustration, in the project that there's as it were one interrupt handler in your extend and calls one isr_evt_set(), the FIFO Queue buffer measure of 4 sections is adequate. In case there are more interrupts utilized within the project that utilize the isr_ communication with RTX kernel or, one interrupt handler that calls a few isr_ library functions, the FIFO Queue buffer estimate should be expanded. The interrupt, that don't utilize isr_ library capacities are not counted here
- Default FIFO Queue buffer measure is 16 passages. This ought to be sufficient for a ordinary RTX project.

- **OS_FIFOSZ** indicates the number of entries that can be put away to the FIFO Queue buffer. Default measure is 16 sections.

`#define OS_FIFOSZ 16`



Bill of materiel:

TIVAC-door lock solenoid -1_channelrelay_5V-KEYPAD

USB cable

Part_1_code_of_project

```
#include "stdint.h"  "MAIN CODE"
#include "C:/Keil/EE319KwareSpring2020/inc/tm4c123gh6pm.h"
#include  "GPIO_int.h"
#include  "kpd_int.h"
#include <string.h> // to use strcmp() function
#define BIT_IS_SET(REG,BIT) ( REG & (1<<BIT) ) // use to define set bit number
#define BIT_IS_CLEAR(REG,BIT) ( !(REG & (1<<BIT)) ) // use to define clear bit number
void SystemInit(){};
        char *ROOMNUM[8]; //  set up here room number 9 room
          //free is mean chickout
        // Occupied is mean chick in
        // Room_Cleaning is mean  Room want to Clean
        char *passcode_keypade=""; //used to  set in passcod enter from keypad
        u8 room(char*passcode,char *status){ // function used to handling the room statues
    // 0-lock
   //oxFF->unlock
        char *strg1="Occupied";
        char *strg2="free";
     char *strg3="RoomCleaning";
                        if(strcmp( status,  strg1)==0){ // check state "Occupied"
                   if( strcmp( passcode,passcode_keypade )==0 ){ // check passcode with passcode
guest entr on keypad
                      currnt=0xFF;//open  door
                               return currnt;
                         }
                 }
                        if(strcmp( status,  strg2)==0){// check state "free"
                                 currnt=0; //locked door
                                 return currnt;

                        }
                        if(strcmp( status,  strg3)==0){ // check state "cleaning"
                                currnt=0xFF; //unlocked door
                                 return currnt;
                                }

               currnt=0x00;
                        return currnt;}
                        void UART_Send_Char(char character){ // function used to  get one char
           while(BIT_IS_SET(UART0_FR_R,5)){};UART0_DR_R=character; // here is polling  on FR
register flag until get one in bit 5 to set data-> char in UART0_DR_R
        }
        char UART_Receive_Char(){  // function used to  set one char
        char data;
                while(BIT_IS_SET(UART0_FR_R,4)){};data=UART0_DR_R;// here is polling  on FR
register flag until get one in bit 4 to get data-> char from UART0_DR_R
        return data;
}

void UART_Send_String(const char *Str )
```

```
{
        u8 i = 0;
        while(Str[i] != '\0')
        {
                UART_Send_Char(Str[i]); // pass char to UART_Send_Char()  from array of char
str
                i++;
        }
}
void UART_Receive_String(char *Str)
{

        u8 i = 0;
        Str[i] = UART_Receive_Char();
        while(Str[i]!= '#') // until user enter # to end entering string
        {
                i++;
                Str[i] = UART_Receive_Char(); // get char from UART_Receive_Char()  and store
it in array of char str
        }
        Str[i] = '\0';
}
 void uart_A(){
   /// first Initialize the UART0 module///
         uint32_t delay; // dummy variable
          SYSCTL_RCGCUART_R|=0x01; // enable clock for uart0
          delay =1;// delay until get connect;
        SYSCTL_RCGCGPIO_R  |=0x23  ;// enable clock for Port A,C,F
        delay =1;// delay until get connect;
 GPIO_init(Port_B); // Initialize for use to send 1 Or 0 for senloid is explain in gpio driver
GPIO_init(Port_D); GPIO_init(Port_C);
  GPIO_PORTA_AFSEL_R=0x03;// enable alternate functions on PA1-0
GPIO_PORTA_PCTL_R=(( GPIO_PORTA_PCTL_R&0x0FFFFF00)+0x11);// configure PA1-0 as UART0
        GPIO_PORTA_AMSEL_R&=~0x03; // to ensure that not analog mode
          GPIO_PORTF_AFSEL_R=0; // not needed  alternate functions as used as gpio
         GPIO_PORTF_PCTL_R=0;
        GPIO_PORTF_AMSEL_R=0;
 GPIO_PORTA_LOCK_R=0x4C4F434B ; // used to unlock port F and (A may not needed) to can read or
write
         GPIO_PORTA_CR_R=0x03;
        GPIO_PORTF_LOCK_R=0x4C4F434B;
        GPIO_PORTF_CR_R=0xFF;
         GPIO_PORTA_DEN_R=0x03; // enable digital I/O on PA1-0
           GPIO_PORTF_DIR_R=0xFF;
        GPIO_PORTF_DEN_R=0xFF;// enable digital I/O on PF
        // configer uart0
             // BRD = 80,000,000 / (16*9600) = 520.8333333
        // UARTFBRD[DIVFRAC] = integer(0.8333333 * 64 + 0.5) = 53
         UART0_CTL_R&=~(1<<0);//disable uart
   UART0_IBRD_R=520;
         UART0_FBRD_R=53;
         UART0_LCRH_R=(0x03<<5);// // 8 bit data, no parity bits, 1 stop bit
          UART0_CC_R=0x0; // Configure the UART clock source by writing to the UARTCC register
```

```
        UART0_CTL_R=(1<<0)|(1<<8)|(1<<9); // (bit 0: enable uart) (bit 8: enable Tx) (bit 9:
enable Rx)

 }

 int main(){
        u8 kpd_sc[3][4];
       uart_A(); //  Initialize and configure uart0
        // setup mode entering number of room
        UART_Receive_String( ROOMNUM[0]);//100#200#*#  in which (100#) string and (200#)
anther  string
        u8 h=0;     ///if he want to end setup mode he must enter* followed by # string
              /// every string is enter must followed by # to end string he enter and to
enter new string
        while(strcmp( ROOMNUM[h], "*")&&h<9){
            h++;
              UART_Receive_String( ROOMNUM[h]);
               }
        //// end setup as  is entre nubmber of room once
       while(1){
               char* status;
               char* status_check;
              char* passcode;
              char*roomnum;
              /*passcode_keypade="1";
                status="Occupied";
                status_check=status;
               passcode="1" ;      //for simulation
               roomnum="1";
              ROOMNUM[0]="1";
       ROOMNUM[1]="2";
       */
       // uart---input
              UART_Receive_String(status);
               UART_Receive_String(roomnum);
                status_check=status;
 if(strcmp( status_check, "Occupied" )==0){ // this we set the passcode only visitor want to
check in
              UART_Receive_String( passcode);
       }

if(strcmp( status_check, "Occupied" )==0){ // here guest need to entre passcode on keypad only
if status "Occupied"
              //keypad function set_in
               if(strcmp( roomnum, ROOMNUM[0])==0){
           passcode_keypade=KPD_getPassword(kpd_sc, Port_D); // keypad for room 1
                       // keypad connceted to portD
                    }
                   if(strcmp( roomnum, ROOMNUM[1])==0){
           passcode_keypade=KPD_getPassword(kpd_sc, Port_C); // keypad for room 2
 // here i must  change  from connceted  keypad portD to  portC as each room have its keypad
and so its port
                    }
```

```
// here i can add many if statement to check room number and many  keypads with many ports
according to max room number and so on
                }
        // here in above code put high (one) on selnoid_door from port b from PB0 toPB7
                if(strcmp( roomnum, ROOMNUM[0])==0){
//GPIO_PORTF_DATA_R= (0x02)&(room(passcode, status)); //for simulation
GPIO_PORTB_DATA_R=(0x01)&(room(passcode, status));//// PB0 is high equal 1 to open
selnoid_Door for room_1
                        }

                 if(strcmp( roomnum, ROOMNUM[1])==0){
//      GPIO_PORTF_DATA_R= (0x04)&(room(passcode, status)); //for simulation
GPIO_PORTB_DATA_R=(0x02)&(room(passcode, status));// PB1 is high equal 1 to open selnoid_Door
for room_2
                        }
        //and so on  i can add many if statement  to check room number to open boor  according
to max room number
                } // end of while(1)
 } // end of main
```

" part of GPIO driver"

```
#define assignBits(var,Bno,val) do{if(val==1)
((var)|= (1 << (Bno))); \
else if(val==0) ((var)&= ~(1 << (Bno))); \
}while(0)
#define getBit(PIN,Pin) (PIN=(PIN & (1<<Pin)))

void GPIO_init(u8 Port){//enable LOCK , enable registers write

            switch(Port){

                case Port_B:
        GPIO_PORTA_LOCK_R = 0x4C4F434B;
GPIO_PORTB_CR_R=0xFF; //enable registers
writing
      GPIO_PORTB_AFSEL_R = 0; //no alternative
functions
      GPIO_PORTB_PCTL_R = 0; // no alternative
functions
      GPIO_PORTB_AMSEL_R = 0; // analog
function disabled
      GPIO_PORTB_DEN_R= 0xFF; // digital
enable
      GPIO_PORTB_PUR_R = 255;
                        break;
```

" Keypad Driver "

```
#define u8unsigned char#define no_Of_Rows 4
#define no_Of_Coloumns 3
u8 INPUT_Pin[no_Of_Rows]={Pin0 , Pin1 ,Pin2 ,
Pin3};
u8 OUTPUT_Pin[no_Of_Coloumns]={Pin4 , Pin5
,Pin6};
voidKPD_ScanKeys(u8KPD_Array[no_Of_Coloumns][no
_Of_Rows],u8 Port_KPD){
      u8 coloumn , row , setter;
 for(coloumn=0 ; coloumn <no_Of_Coloumns ;
coloumn++){
```

```
case Port_c: "GPIO driver complete"

        GPIO_PORTC_LOCK_R = 0x4C4F434B;
      GPIO_PORTC_CR_R=0xFF; //enable
registers writing
      GPIO_PORTC_AFSEL_R = 0; //no
alternative functions
      GPIO_PORTC_PCTL_R = 0; // no
alternative functions
      GPIO_PORTC_AMSEL_R = 0; // analog
function disabled
      GPIO_PORTC_DEN_R= 0xFF; // digital
enable
     GPIO_PORTC_PUR_R = 255;
                break;
     }
```

" Keypad Driver complete "

```
for(setter=0 ; setter < no_Of_Coloumns ;
setter++){

if(setter==coloumn)
GPIO_setPinValue(Port_KPD,OUTPUT_Pin[setter
] ,LOW);
        else GPIO_setPinValue(Port_KPD
,OUTPUT_Pin[setter] ,HIGH);}
for(row=0 ; row < no_Of_Rows ; row++){
u8 level;
GPIO_getPinValue(Port_KPD,INPUT_Pin[row],&l
evel);
        KPD_Array[coloumn][row] = level;
                        }
                }// end of first "for"
}// end of KPD_ScanKeys function
```

```
 "   Keypad Driver complete "
short int KPD_getNumber(u8
KPD_Array[no_Of_Coloumns][no_Of_Rows]){
if(KPD_Array[0][0]==0) return 1;
        if(KPD_Array[1][0]==0) return 2;
        if(KPD_Array[2][0]==0) return 3;
        if(KPD_Array[0][1]==0) return 4;
        if(KPD_Array[1][1]==0) return 5;
        if(KPD_Array[2][1]==0) return 6;
        if(KPD_Array[0][2]==0) return 7;
        if(KPD_Array[1][2]==0) return 8;
        if(KPD_Array[2][2]==0) return 9;
        if(KPD_Array[0][3]==0) return '*';
if(KPD_Array[1][3]==0) return0;return -1;}
 char* KPD_getPassword(u8
KPD_Array[no_Of_Coloumns][no_Of_Rows], u8
Port_KPD){
u8 numberOfKPDInputs=0;
        short int value;
        char*enteredPassword;
while(numberOfKPDInputs!=4){
KPD_ScanKeys(KPD_Array, Port_KPD);
value=KPD_getNumber(KPD_Array);
        if(value == -1) //this means user
doesn't press any button
        continue;
enteredPassword[numberOfKPDInputs]=(char)val
ue;
        numberOfKPDInputs++;
                }
        return enteredPassword;}
```

3-TI_RTOS

http://www.ti.com/lit/ug/spruhd4m/spruhd4m.pdf?ts=1590994563834

-http://www.ti.com/lit/ug/spruex3t/spruex3t.pdf

http://www.ti.com/lit/an/sprabw1/sprabw1.pdf?ts=1591109625187

4-Link for code on GITHUB:
https://github.com/midoamrm/micro_TIva_C_hotelroom_project .

References

1-Free-RTOS:

-https://www.freertos.org/wp-content/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf

-https://www.freertos.org/RTOS.html

-https://www.freertos.org/wp-content/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf

2-Keil-RTOS

-http://www.keil.com/support/man/docs/rlarm/rlarm_ar_artxarm.htm

-http://www.keil.com/pack/doc/CMSIS_Dev/RTOS2/html/index.html