Федеральное государственное бюджетное образовательное учреждение
высшего образования

«Московский государственный технический университет
им. Н.Э. Баумана (национальный исследовательский
университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ – Информатика, искусственный интеллект и системы управления
КАФЕДРА – Информационные системы и телекоммуникации

Отчет к лабораторной работе №2

по курсу "Программное обеспечение встроенных систем"

направления 09.04.02

"Проектирование с применением ядра Nios II и ОСРВ uC/OS-II"

Выполнил:

студент группы ИУ3-11М

Щесняк С.С.

Проверил:

Федоров С.В.

Москва 2022

Оглавление

1. Цель работы

Освоить практические навыки проектирования программного обеспечения (ПО) для операционных систем реального времени (ОСРВ) при создании простого проекта программного обеспечения системы на кристалле Nios II на основе ОСРВ uC/OS-II. Изучить основные сервисы ОСРВ uC/OS-II. Реализовать обработчик прерывания в системе на кристалле Nios II. Освоить методики обработки внешних событий в uC/OS-II.

2. Задание

1. Реализуйте в задаче print_status_task вывод информации о времени начала очередного цикла выполнения задачи после вызова функции ожидания в консольное окно и о количестве сообщений в очереди. Объясните, почему в очереди столько сообщений.
2. В наименее приоритетную задачу getsem_task2 добавьте локальную 8-ми разрядную переменную INT8U led и измените цикл следующим образом (добавленные строки отмечены синим курсивом):

   ...
   INT8U led;
   led=1;
   while (1)
   {
   led = led<<1;
   if (led==0) led=1;
   IOWR_ALTERA_AVALON_PIO_DATA(GREEN_LED_BASE,led);
   OSSemPend(...
   ...
3. По файлу .objdump оцените объем памяти программ и данных, который требуется с настройками ОС по умолчанию.
4. Требуется создать проект на основе шаблона, модифицировав его следующим образом:

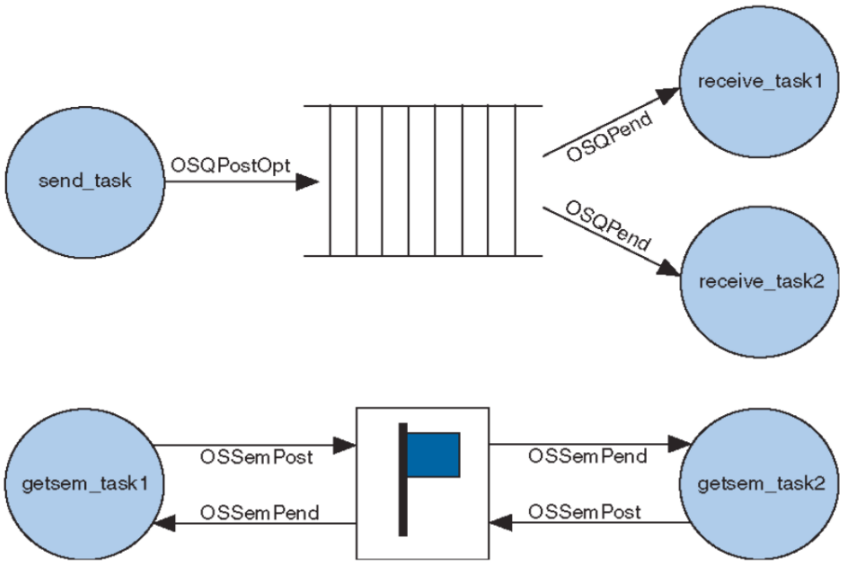
a. Нажатие на кнопку на отладочной плате должно вызывать обработчик прерывания, который посылает сообщение о нажатой кнопке в задачу send_task.
b. В отличие от шаблона, задача send_task должна посылать сообщения не по таймауту, а при получении сообщения от прерывания. При этом ожидание в задачах send_task, receive_task1 и receive_task2 следует убрать - задачи будут выполняться при нажатии кнопки.
c. Реализуйте широковещательную рассылку сообщения о нажатии кнопки одновременно двум ждущим задачам receive_task.
d. Задача receive_task1 должна выводить код нажатой кнопки на левый семисегментный индикатор.
e. Задача receive_task2 должна выводить код нажатой кнопки на правый семисегментный индикатор.
f. Так как задачи receive_task1 и receive_task используют разделяемый ресурс - порт вывода на семисегментные индикаторы, следует реализовать контроль доступа к этому ресурсу.

5. Модифицируйте программу в соответствии с индивидуальным заданием. Проведите тестирование программы. Сохраните в отчет информацию об объеме данных и кода.
    a. **Поворотник.** При нажатии левой кнопки происходит следующее: зажигается правый светодиод, через 50мс находящийся левее от него и т.д. до включения всех светодиодов. Потом светодиоды гаснут и не горят 100мс. Процесс повторяется три раза. При нажатии правой кнопки аналогично, но процесс начинается с левого светодиода. При нажатии любой кнопки в процессе моргания переключение направления осуществляется после гашения и истечения интервала 100мс, не дожидаясь завершения трех итераций.

6. В BSP Editor отключите неиспользуемые сервисы ОС, приведите количество объектов и задач в настройке параметров ОС в соответствие требованиям задачи. Проведите повторное тестирование программы. Сравните объем данных и кода. Занесите результаты в отчет.

3. Ход работы

1. Структура приложения, реализуемого на этом этапе выполнения лабораторной работы, приведена на рис. 1. В приложении реализованы пять задач. Задача send_task заполняет очередь сообщений увеличивающимся значением. Задачи receive_task1 и receive_task2 периодически выбирают из очереди сообщения. Задачи getsem_task1 и getsem_task2 осуществляют доступ к общему ресурсу, который защищен семафором. Также в проекте используются две непоказанные на рисунке служебные задачи – одна для создания других задач после



запуска ОС и одна для вывода сообщений в консоль.

Рисунок 1. Структура первоначального приложения

Листинг кода:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
```

```
#include "includes.h"
#include "alt_ucosii_simple_error_check.h"
#include "altera_avalon_pio_regs.h"

/* Initialize stacks for tasks */
#define   TASK_STACKSIZE       1024
OS_STK    initialize_task_stk[TASK_STACKSIZE];
OS_STK    print_status_task_stk[TASK_STACKSIZE];
OS_STK    getsem_task1_stk[TASK_STACKSIZE];
OS_STK    getsem_task2_stk[TASK_STACKSIZE];
OS_STK    receive_task1_stk[TASK_STACKSIZE];
OS_STK    receive_task2_stk[TASK_STACKSIZE];
OS_STK    send_task_stk[TASK_STACKSIZE];

/* Define task priorities - the less the number, the more priority it
has */
#define INITIALIZE_TASK_PRIORITY   6
#define PRINT_STATUS_TASK_PRIORITY 7
#define GETSEM_TASK1_PRIORITY      8
#define GETSEM_TASK2_PRIORITY      9
#define RECEIVE_TASK1_PRIORITY    10
#define RECEIVE_TASK2_PRIORITY    11
#define SEND_TASK_PRIORITY        12


/* Declare queue for send task and receive tasks to use to pass
messages */
#define   MSG_QUEUE_SIZE  30            //Size of message queue used in
example
OS_Q_DATA message_data;                   //Data structure used to hold
queue data
OS_EVENT  *msgqueue;                     //Message queue pointer
void      *msgqueueTbl[MSG_QUEUE_SIZE]; // Storage for messages


/* Declare semaphore for getsem tasks to use to gain access to a shared
resource */
OS_EVENT *shared_resource_sem;

/* Global variables */
INT32U number_of_messages_sent = 0;
INT32U number_of_messages_received_task1 = 0;
```

```
INT32U number_of_messages_received_task2 = 0;
INT32U getsem_task1_got_sem = 0;
INT32U getsem_task2_got_sem = 0;
char sem_owner_task_name[20];

/* Function prototypes */
int initOSDataStructs(void);
int initCreateTasks(void);

/* Service task used to print information about how this program functions */
void print_status_task(void* pdata)
{
  while (1)
  {
    OSTimeDlyHMSM(0, 0, 3, 0); //Sleeping for 3 seconds
    /* Getting the amount of time that had passed since the programm started working (in ticks of system clock)
     * using OSTimeGet() and converting it to seconds, minutes and hours. OS_TICKS_PER_SEC = 1000 is a pre-defined
     * constant (in system.h) that tells us how many ticks of the system clock there are in a second. */
    int timeSinceStartS = (OSTimeGet()/OS_TICKS_PER_SEC)%60; //Getting the amount of leftover seconds from the time that had passed since the start of the system timer
    int timeSinceStartM = (OSTimeGet()/(OS_TICKS_PER_SEC*60))%60; //Getting the amount of leftover minutes from the time that had passed since the start of the system timer
    int timeSinceStartH = (OSTimeGet()/(OS_TICKS_PER_SEC*3600))%60; //Getting the amount of leftover hours from the time that had passed since the start of the system timer
    INT8U err = OSQQuery(msgqueue, &message_data); //Get information about msgqueue in order to determine the amount of messages in queue

printf("*********************************************************************\n");
    printf("Hello From MicroC/OS-II Running on NIOS II.  Here is the status:\n");
    printf("\n");
    printf("The number of messages sent by the send_task: %lu\n",
            number_of_messages_sent);
```

```
    printf("\n");
    printf("The number of messages received by the receive_task1:
%lu\n",
            number_of_messages_received_task1);
    printf("\n");
    printf("The number of messages received by the receive_task2:
%lu\n",
            number_of_messages_received_task2);
    printf("\n");
    printf("The shared resource is owned by: %s\n",
           &sem_owner_task_name[0]);
    printf("\n");
    printf("The Number of times getsem_task1 acquired the semaphore
%lu\n",
            getsem_task1_got_sem);
    printf("\n");
    printf("The Number of times getsem_task2 acquired the semaphore
%lu\n",
            getsem_task2_got_sem);
    printf("\n");
    printf("Time                    since                    launch:
%d:%d:%d\n",timeSinceStartH,timeSinceStartM,timeSinceStartS);    //print
time since
    printf("\n");
    if(err==OS_ERR_NONE){ //If information about msgqueue was obtained,
print the number of messages in queue
         printf("The      Number      of      messages:      %lu\n",
(long)message_data.OSNMsgs);
         printf("\n");
    }

printf("************************************************************
*\n");
    printf("\n");
  }
}

/* Task that is used to gain access to a shared resource */
void getsem_task1(void* pdata)
{
  INT8U return_code = OS_NO_ERR;
```

```
  while (1)
  {
    OSSemPend(shared_resource_sem, 0, &return_code); //Wait until the
semaphore is set to 1 (without a timeout), then lock the semaphore
    alt_ucosii_check_return_code(return_code); //Check the return code
for any errors present
    strcpy(&sem_owner_task_name[0], "getsem_task1"); //Specifies that
the shared resource is now owned by this task by writing to a global
variable
    getsem_task1_got_sem++; //Increase the number of times this task
had access to the shared resource
    OSSemPost(shared_resource_sem); //Release the semaphore
    OSTimeDlyHMSM(0, 0, 0, 100); //Sleep for 100ms
  }
}

/* Task that is used to gain access to a shared resource and to create
a running light*/
void getsem_task2(void* pdata)
{
  INT8U return_code = OS_NO_ERR;
  INT8U led; //Stores the current position of the LED that is switched
on
  led=2; //Sets the starting position to the second-right LED
  while (1)
  {
 led = led << 1; //Move the position of the LED that is switched on 1
to the left
 if(led==0) led=2; //If we were at the very left of the LED array,
reset the LED position
         IOWR_ALTERA_AVALON_PIO_DATA(GREEN_LED_BASE,led); //Write the
position of the switched-on LED to the address defined by
GREEN_LED_BASE
    OSSemPend(shared_resource_sem, 0, &return_code); //Same as
getsem_task1, except here we sleep for 130ms
    strcpy(&sem_owner_task_name[0], "getsem_task2");
    alt_ucosii_check_return_code(return_code);
    getsem_task2_got_sem++;
    OSSemPost(shared_resource_sem);
    OSTimeDlyHMSM(0, 0, 0, 130);
  }
}
```

```
/* Task that is used to send the current message number to
receive_task1 and receive_task2 */
void send_task(void* pdata)
{
  INT8U return_code = OS_NO_ERR;
  INT32U  msg = 0;
  OS_Q_DATA queue_data;

  while (1)
  {
    return_code = OSQQuery(msgqueue, &queue_data); //Get information
about msgqueue in order to determine the amount of messages in queue
    alt_ucosii_check_return_code(return_code);
    if(queue_data.OSNMsgs < MSG_QUEUE_SIZE) //If the message queue
isn't overflowing, send a message to all awaiting tasks
    {
      return_code     =     OSQPostOpt(msgqueue,     (void     *)&msg,
OS_POST_OPT_BROADCAST); //Send the current message number to all
awaiting tasks through msgqueue
      alt_ucosii_check_return_code(return_code);
      msg++; //Increment the current message number
      number_of_messages_sent++; //Increase the number of times a
message was sent by this task
    }
    else
    {
      OSTimeDlyHMSM(0, 0, 1, 0); //Sleep for 1s
    }
  }
}

/* Task that is used to listen to and read messages from msgqueue */
void receive_task1(void* pdata)
{
  INT8U return_code = OS_NO_ERR;
  INT32U *msg;

  while (1)
  {
    msg = (INT32U *)OSQPend(msgqueue, 0, &return_code); //Wait for a
message to appear in msgqueue, then retrieve said message
```

```
    alt_ucosii_check_return_code(return_code); //Check the return code
for any errors present
    number_of_messages_received_task1++; //Increase the amount of times
this task had received and handled a message from msgqueue
    OSTimeDlyHMSM(0, 0, 0, 333); //Sleep for 1/3 of a second
  }
}

/* Task that is used to listen to and read messages from msgqueue */
void receive_task2(void* pdata)
{
  INT8U return_code = OS_NO_ERR;
  INT32U *msg;

  while (1)
  {
    msg = (INT32U *)OSQPend(msgqueue, 0, &return_code); //Same as
receive_task1
    alt_ucosii_check_return_code(return_code);
    number_of_messages_received_task2++;
    OSTimeDlyHMSM(0, 0, 1, 0);
  }
}

/* Service task that is used to create other tasks and initialize OS
data structures, such as message queues, mailboxes, semaphores and
mutexes */
void  initialize_task(void* pdata)
{
  INT8U return_code = OS_NO_ERR;

  initOSDataStructs();

  initCreateTasks();

  return_code = OSTaskDel(OS_PRIO_SELF);
  alt_ucosii_check_return_code(return_code);
  while (1);
}

/* Starting program execution. Creating initialize_task and starting OS
*/
```

```
int main (int argc, char* argv[], char* envp[])
{
  INT8U return_code = OS_NO_ERR;
  return_code = OSTaskCreate(initialize_task,
                             NULL,
                             (void
*)&initialize_task_stk[TASK_STACKSIZE-1],
                             INITIALIZE_TASK_PRIORITY);
  alt_ucosii_check_return_code(return_code);
  OSStart();
  return 0;
}

/* Initializing OS data structures - a queue and a semaphore */
int initOSDataStructs(void)
{
  msgqueue = OSQCreate(&msgqueueTbl[0], MSG_QUEUE_SIZE);
  shared_resource_sem = OSSemCreate(1);
  return 0;
}

/* Creating all tasks described above (except initialize_task) */
int initCreateTasks(void)
{
  INT8U return_code = OS_NO_ERR;

  return_code = OSTaskCreate(getsem_task1,
                             NULL,
                             (void   *)&getsem_task1_stk[TASK_STACKSIZE-
1],
                             GETSEM_TASK1_PRIORITY);
  return_code = OSTaskCreate(getsem_task2,
                             NULL,
                             (void   *)&getsem_task2_stk[TASK_STACKSIZE-
1],
                             GETSEM_TASK2_PRIORITY);
  return_code = OSTaskCreate(receive_task1,
                             NULL,
                             (void  *)&receive_task1_stk[TASK_STACKSIZE-
1],
                             RECEIVE_TASK1_PRIORITY);
  alt_ucosii_check_return_code(return_code);
```

```
  return_code = OSTaskCreate(receive_task2,
                             NULL,
                             (void  *)&receive_task2_stk[TASK_STACKSIZE-
1],
                             RECEIVE_TASK2_PRIORITY);
  alt_ucosii_check_return_code(return_code);
  return_code = OSTaskCreate(send_task,
                             NULL,
                             (void *)&send_task_stk[TASK_STACKSIZE-1],
                             SEND_TASK_PRIORITY);
  alt_ucosii_check_return_code(return_code);
  return_code = OSTaskCreate(print_status_task,
                             NULL,
                             (void
*)&print_status_task_stk[TASK_STACKSIZE-1],
                             PRINT_STATUS_TASK_PRIORITY);
  alt_ucosii_check_return_code(return_code);
  return 0;
}
```

Анализ исходного кода показывает, что задержки в задачах send_task, receive_task1 и receive_task2 подобраны так, что между стартом ожидания задачей send_task обработки добавленных в очередь сообщений и его окончанием задачи receive_task1 и receive_task2 успевают обработать три сообщения. Затем (раз в три цикла работы данной части программы) задача print_task, которая обладает более высоким приоритетом, чем задача send_task, выводит в консоль число сообщений в очереди в данный момент: 27. Наконец, за время, меньшее времени ожидания задач receive_task1 и receive_task2 очередь снова заполняется, в связи с чем задаче send_task снова приходится ожидать обработки отправленных сообщений и вывода числа оставшихся после обработки.

2. Листинг кода, написанного в рамках выполнения данного этапа лабораторной работы, представлен выше.

3. Вид файла .objdump, , полученного в рамках выполнения данного этапа лабораторной работы представлен на рисунке 3

```
ucosii_tutorial.elf:     file format elf32-littlenios2
ucosii_tutorial.elf
architecture: nios2:r1, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00000238

Program Header:
    LOAD off    0x00001000 vaddr 0x00000000 paddr 0x00000000 align 2**12
         filesz 0x00000020 memsz 0x00000020 flags r-x
    LOAD off    0x00001020 vaddr 0x00000020 paddr 0x00000020 align 2**12
         filesz 0x00019844 memsz 0x00019844 flags r-x
    LOAD off    0x0001a864 vaddr 0x00019864 paddr 0x0001b4c4 align 2**12
         filesz 0x00001c60 memsz 0x00001c60 flags rw-
    LOAD off    0x0001d124 vaddr 0x0001d124 paddr 0x0001d124 align 2**12
         filesz 0x00000000 memsz 0x0000a48c flags rw-

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .entry        00000020  00000000  00000000  00001000  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .exceptions   00000218  00000020  00000020  00001020  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .text         00018928  00000238  00000238  00001238  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .rodata       00000d04  00018b60  00018b60  00019b60  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .rwdata       00001c60  00019864  0001b4c4  0001a864  2**2
                  CONTENTS, ALLOC, LOAD, DATA, SMALL_DATA
  5 .bss          0000a48c  0001d124  0001d124  0001d124  2**2
                  ALLOC, SMALL_DATA
```

Рисунок 3. Вид файла .objdump, полученного в рамках выполнения третьего этапа лабораторной работы

Анализ представленных в данном файле сведений показывает, что объем кода составляет 98 Кбайт, а объем неинициализированных данных – 41 Кбайт.

4. Структура приложения, реализуемого на этом этапе выполнения лабораторной работы, приведена на рис. 4. В приложении реализованы четыре задачи. Задача send_task выбирает из очереди btnqueue, заполняемой в соответствующем обработчике прерывания, сообщения с кодами нажатых кнопок и пересылает их другим задачам, заполняя очереди msgqueue и indqueue. Задачи receive_task1 и receive_task2 периодически выбирают из очереди msgqueue сообщения и осуществляют доступ к общему ресурсу (семисегментному

индикатору), который защищен семафором. Также в проекте используются непоказанная на рисунке служебная задача, предназначенная для создания других задач после запуска ОС.

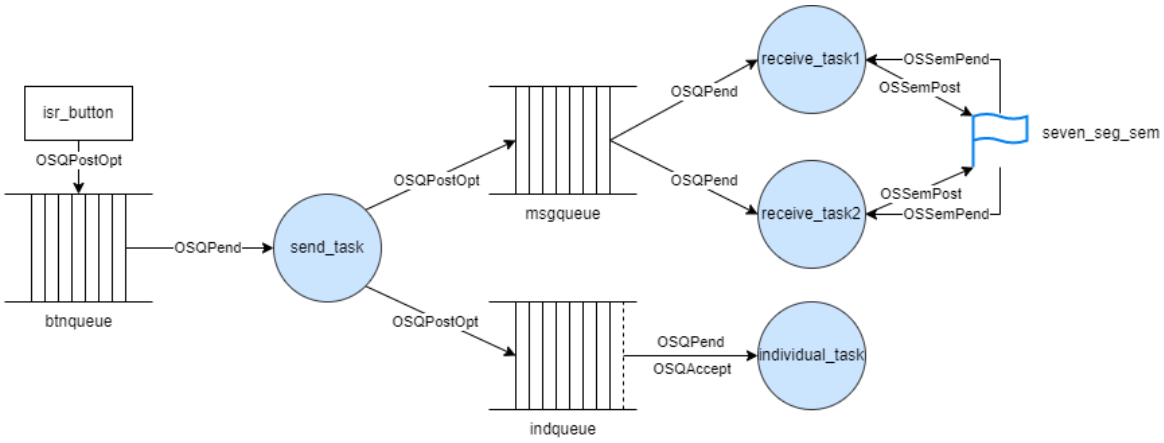


Рисунок 4. Структура модифицированного приложения

Листинг кода:

```
#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include "includes.h"

#include "alt_ucosii_simple_error_check.h"

#include "altera_avalon_pio_regs.h"


/* Initialize stacks for tasks */

#define   TASK_STACKSIZE       1024

OS_STK    initialize_task_stk[TASK_STACKSIZE];

OS_STK    print_status_task_stk[TASK_STACKSIZE];

OS_STK    getsem_task1_stk[TASK_STACKSIZE];

OS_STK    getsem_task2_stk[TASK_STACKSIZE];

OS_STK    receive_task1_stk[TASK_STACKSIZE];

OS_STK    receive_task2_stk[TASK_STACKSIZE];

OS_STK    send_task_stk[TASK_STACKSIZE];
```

```
/* Define task priorities - the less the number, the more priority it
has */

#define INITIALIZE_TASK_PRIORITY   6

#define PRINT_STATUS_TASK_PRIORITY 7

#define GETSEM_TASK1_PRIORITY      8

#define GETSEM_TASK2_PRIORITY      9

#define RECEIVE_TASK1_PRIORITY    10

#define RECEIVE_TASK2_PRIORITY    11

#define SEND_TASK_PRIORITY        12



/* Declare queue for send task and receive tasks to use to pass messages
*/

#define   MSG_QUEUE_SIZE  30            //Size of message queue used in
example

OS_Q_DATA message_data;                   //Data structure used to hold
queue data

OS_EVENT  *msgqueue;                     //Message queue pointer

void      *msgqueueTbl[MSG_QUEUE_SIZE]; // Storage for messages



/* Declare semaphore for getsem tasks to use to gain access to a shared
resource */

OS_EVENT *shared_resource_sem;


/* Global variables */

INT32U number_of_messages_sent = 0;

INT32U number_of_messages_received_task1 = 0;
```

```
INT32U number_of_messages_received_task2 = 0;

INT32U getsem_task1_got_sem = 0;

INT32U getsem_task2_got_sem = 0;

char sem_owner_task_name[20];


/* Function prototypes */

int initOSDataStructs(void);

int initCreateTasks(void);


/* Service task used to print information about how this program functions */

void print_status_task(void* pdata)

{

  while (1)

  {

    OSTimeDlyHMSM(0, 0, 3, 0); //Sleeping for 3 seconds

    /* Getting the amount of time that had passed since the programm started working (in ticks of system clock)

     * using OSTimeGet() and converting it to seconds, minutes and hours. OS_TICKS_PER_SEC = 1000 is a pre-defined

     * constant (in system.h) that tells us how many ticks of the system clock there are in a second. */

    int timeSinceStartS = (OSTimeGet()/OS_TICKS_PER_SEC)%60; //Getting the amount of leftover seconds from the time that had passed since the start of the system timer

    int timeSinceStartM = (OSTimeGet()/(OS_TICKS_PER_SEC*60))%60; //Getting the amount of leftover minutes from the time that had passed since the start of the system timer

    int timeSinceStartH = (OSTimeGet()/(OS_TICKS_PER_SEC*3600))%60; //Getting the amount of leftover hours from the time that had passed since the start of the system timer
```

```
    INT8U err = OSQQuery(msgqueue, &message_data); //Get information
about msgqueue in order to determine the amount of messages in queue


printf("************************************************************
\n");

    printf("Hello From MicroC/OS-II Running on NIOS II.  Here is the
status:\n");

    printf("\n");

    printf("The number of messages sent by the send_task:
%lu\n",

            number_of_messages_sent);

    printf("\n");

    printf("The number of messages received by the receive_task1:
%lu\n",

            number_of_messages_received_task1);

    printf("\n");

    printf("The number of messages received by the receive_task2:
%lu\n",

            number_of_messages_received_task2);

    printf("\n");

    printf("The shared resource is owned by: %s\n",

           &sem_owner_task_name[0]);

    printf("\n");

    printf("The Number of times getsem_task1 acquired the semaphore
%lu\n",

            getsem_task1_got_sem);

    printf("\n");

    printf("The Number of times getsem_task2 acquired the semaphore
%lu\n",

            getsem_task2_got_sem);
```

```
    printf("\n");

    printf("Time since launch: %d:%d:%d\n",timeSinceStartH,timeSinceStartM,timeSinceStartS);    //print time since

    printf("\n");

    if(err==OS_ERR_NONE){ //If information about msgqueue was obtained, print the number of messages in queue

      printf("The Number of messages: %lu\n", (long)message_data.OSNMsgs);

      printf("\n");

    }


printf("*****************************************************************\n");

    printf("\n");

  }

}


/* Task that is used to gain access to a shared resource */

void getsem_task1(void* pdata)

{

  INT8U return_code = OS_NO_ERR;


  while (1)

  {

    OSSemPend(shared_resource_sem, 0, &return_code); //Wait until the semaphore is set to 1 (without a timeout), then lock the semaphore

    alt_ucosii_check_return_code(return_code); //Check the return code for any errors present
```

```
    strcpy(&sem_owner_task_name[0], "getsem_task1"); //Specifies that the shared resource is now owned by this task by writing to a global variable

    getsem_task1_got_sem++; //Increase the number of times this task had access to the shared resource

    OSSemPost(shared_resource_sem); //Release the semaphore

    OSTimeDlyHMSM(0, 0, 0, 100); //Sleep for 100ms

  }

}


/* Task that is used to gain access to a shared resource and to create a running light*/

void getsem_task2(void* pdata)

{

  INT8U return_code = OS_NO_ERR;

  INT8U led; //Stores the current position of the LED that is switched on

  led=2; //Sets the starting position to the second-right LED

  while (1)

  {

      led = led << 1; //Move the position of the LED that is switched on 1 to the left

      if(led==0) led=2; //If we were at the very left of the LED array, reset the LED position

            IOWR_ALTERA_AVALON_PIO_DATA(GREEN_LED_BASE,led); //Write the position of the switched-on LED to the address defined by GREEN_LED_BASE

    OSSemPend(shared_resource_sem, 0, &return_code); //Same as getsem_task1, except here we sleep for 130ms

    strcpy(&sem_owner_task_name[0], "getsem_task2");

    alt_ucosii_check_return_code(return_code);
```

```
    getsem_task2_got_sem++;
    OSSemPost(shared_resource_sem);
    OSTimeDlyHMSM(0, 0, 0, 130);
  }
}

/* Task that is used to send the current message number to receive_task1
and receive_task2 */
void send_task(void* pdata)
{
  INT8U return_code = OS_NO_ERR;
  INT32U  msg = 0;
  OS_Q_DATA queue_data;

  while (1)
  {
    return_code = OSQQuery(msgqueue, &queue_data); //Get information
about msgqueue in order to determine the amount of messages in queue
    alt_ucosii_check_return_code(return_code);
    if(queue_data.OSNMsgs < MSG_QUEUE_SIZE) //If the message queue isn't
overflowing, send a message to all awaiting tasks
    {
      return_code     =     OSQPostOpt(msgqueue,     (void     *)&msg,
OS_POST_OPT_BROADCAST); //Send the current message number to all
awaiting tasks through msgqueue
      alt_ucosii_check_return_code(return_code);
      msg++; //Increment the current message number
      number_of_messages_sent++; //Increase the number of times a
message was sent by this task
```

```
    }
    else
    {
      OSTimeDlyHMSM(0, 0, 1, 0); //Sleep for 1s
    }
  }
}

/* Task that is used to listen to and read messages from msgqueue */
void receive_task1(void* pdata)
{
  INT8U return_code = OS_NO_ERR;
  INT32U *msg;

  while (1)
  {
    msg = (INT32U *)OSQPend(msgqueue, 0, &return_code); //Wait for a message to appear in msgqueue, then retrieve said message
    alt_ucosii_check_return_code(return_code); //Check the return code for any errors present
    number_of_messages_received_task1++; //Increase the amount of times this task had received and handled a message from msgqueue
    OSTimeDlyHMSM(0, 0, 0, 333); //Sleep for 1/3 of a second
  }
}

/* Task that is used to listen to and read messages from msgqueue */
void receive_task2(void* pdata)
```

```
{
  INT8U return_code = OS_NO_ERR;
  INT32U *msg;

  while (1)
  {
    msg = (INT32U *)OSQPend(msgqueue, 0, &return_code); //Same as receive_task1
    alt_ucosii_check_return_code(return_code);
    number_of_messages_received_task2++;
    OSTimeDlyHMSM(0, 0, 1, 0);
  }
}

/* Service task that is used to create other tasks and initialize OS data structures, such as message queues, mailboxes, semaphores and mutexes */
void  initialize_task(void* pdata)
{
  INT8U return_code = OS_NO_ERR;

  initOSDataStructs();

  initCreateTasks();

  return_code = OSTaskDel(OS_PRIO_SELF);
  alt_ucosii_check_return_code(return_code);
  while (1);
```

```
}


/* Starting program execution. Creating initialize_task and starting OS
*/
int main (int argc, char* argv[], char* envp[])
{
  INT8U return_code = OS_NO_ERR;
  return_code = OSTaskCreate(initialize_task,
                             NULL,
                             (void
*)&initialize_task_stk[TASK_STACKSIZE-1],
                             INITIALIZE_TASK_PRIORITY);
  alt_ucosii_check_return_code(return_code);
  OSStart();
  return 0;
}


/* Initializing OS data structures - a queue and a semaphore */
int initOSDataStructs(void)
{
  msgqueue = OSQCreate(&msgqueueTbl[0], MSG_QUEUE_SIZE);
  shared_resource_sem = OSSemCreate(1);
  return 0;
}


/* Creating all tasks described above (except initialize_task) */
int initCreateTasks(void)
{
```

```
  INT8U return_code = OS_NO_ERR;


  return_code = OSTaskCreate(getsem_task1,
                             NULL,
                             (void   *)&getsem_task1_stk[TASK_STACKSIZE-
1],
                             GETSEM_TASK1_PRIORITY);
  return_code = OSTaskCreate(getsem_task2,
                             NULL,
                             (void   *)&getsem_task2_stk[TASK_STACKSIZE-
1],
                             GETSEM_TASK2_PRIORITY);
  return_code = OSTaskCreate(receive_task1,
                             NULL,
                             (void  *)&receive_task1_stk[TASK_STACKSIZE-
1],
                             RECEIVE_TASK1_PRIORITY);
  alt_ucosii_check_return_code(return_code);
  return_code = OSTaskCreate(receive_task2,
                             NULL,
                             (void  *)&receive_task2_stk[TASK_STACKSIZE-
1],
                             RECEIVE_TASK2_PRIORITY);
  alt_ucosii_check_return_code(return_code);
  return_code = OSTaskCreate(send_task,
                             NULL,
                             (void *)&send_task_stk[TASK_STACKSIZE-1],
                             SEND_TASK_PRIORITY);
```

```
  alt_ucosii_check_return_code(return_code);

  return_code = OSTaskCreate(print_status_task,

                             NULL,

                             (void
*)&print_status_task_stk[TASK_STACKSIZE-1],

                             PRINT_STATUS_TASK_PRIORITY);

  alt_ucosii_check_return_code(return_code);

  return 0;

}
```

5. Вид файла .objdump, , полученного в рамках выполнения данного этапа лабораторной работы представлен на рисунке 5

```
Program Header:
    LOAD off    0x00001000 vaddr 0x00000000 paddr 0x00000000 align 2**12
         filesz 0x00000020 memsz 0x00000020 flags r-x
    LOAD off    0x00001020 vaddr 0x00000020 paddr 0x00000020 align 2**12
         filesz 0x0001a5a8 memsz 0x0001a5a8 flags r-x
    LOAD off    0x0001b5c8 vaddr 0x0001a5c8 paddr 0x0001c238 align 2**12
         filesz 0x00001c70 memsz 0x00001c70 flags rw-
    LOAD off    0x0001dea8 vaddr 0x0001dea8 paddr 0x0001dea8 align 2**12
         filesz 0x00000000 memsz 0x00008ff0 flags rw-

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .entry        00000020  00000000  00000000  00001000  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .exceptions   00000218  00000020  00000020  00001020  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .text         00019878  00000238  00000238  00001238  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .rodata       00000b18  00019ab0  00019ab0  0001aab0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .rwdata       00001c70  0001a5c8  0001c238  0001b5c8  2**2
                  CONTENTS, ALLOC, LOAD, DATA, SMALL_DATA
  5 .bss          00008ff0  0001dea8  0001dea8  0001dea8  2**2
                  ALLOC, SMALL_DATA
```

Рисунок 5. Вид файла .objdump, полученного в рамках выполнения третьего этапа лабораторной работы

Анализ представленных в данном файле сведений показывает, что первоначальный объем кода модифицированной программы составляет 102 Кбайт, а объем неинициализированных данных – 36 Кбайт.

6. Вид файла .objdump, , полученного в рамках выполнения данного этапа лабораторной работы представлен на рисунке 6

```
Program Header:
    LOAD off    0x00001000 vaddr 0x00000000 paddr 0x00000000 align 2**12
         filesz 0x00000020 memsz 0x00000020 flags r-x
    LOAD off    0x00001020 vaddr 0x00000020 paddr 0x00000020 align 2**12
         filesz 0x00019350 memsz 0x00019350 flags r-x
    LOAD off    0x0001a370 vaddr 0x00019370 paddr 0x0001afe0 align 2**12
         filesz 0x00001c70 memsz 0x00001c70 flags rw-
    LOAD off    0x0001cc50 vaddr 0x0001cc50 paddr 0x0001cc50 align 2**12
         filesz 0x00000000 memsz 0x0000839c flags rw-

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .entry        00000020  00000000  00000000  00001000  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .exceptions   00000218  00000020  00000020  00001020  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .text         00018658  00000238  00000238  00001238  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .rodata       00000ae0  00018890  00018890  00019890  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .rwdata       00001c70  00019370  0001afe0  0001a370  2**2
                  CONTENTS, ALLOC, LOAD, DATA, SMALL_DATA
  5 .bss          0000839c  0001cc50  0001cc50  0001cc50  2**2
                  ALLOC, SMALL_DATA
```

Рисунок 6. Вид файла .objdump, полученного в рамках выполнения третьего этапа лабораторной работы

Анализ представленных в данном файле сведений показывает, что при отключении неиспользуемых сервисов операционной системы объем кода уменьшается до 98 Кбайт, а объем неинициализированных данных – до 33 Кбайт.

4. Вывод

В результате выполнения лабораторной работы при создании простого проекта программного обеспечения системы на кристалле Nios II на основе ОСРВ uC/OS-II были освоены практические навыки проектирования программного обеспечения (ПО) для операционных систем реального времени (ОСРВ), изучены основные сервисы ОСРВ uC/OS-II. Кроме того, были освоены методики обработки внешних событий в uC/OS-II и реализован обработчик прерывания в системе на кристалле Nios II.