**Journal**

**Exercise 1 HW/SW Co-design**

Date: **06/11-10**

Author: **Anders Hvidgaard Poder**

Stud.nr.: 19951439

# Introduction

I decided to try and not use the hints or solutions (other than the ones found in the exercise itself). If this proves impossible it will be stated in the assignment where I had to “cheat”.

# Assignment 1

Followed the exercise text to the letter no problem

# Assignment 2

Followed the exercise text to the letter no problem

# Assignment 3

I created a new project (UART\_cmd\_line) based on the BSP created in Assignment 2.

I wrote the command language interpreter no problem. I have written many of those in my past. I did hit some snags though:

* scanf did not work properly with \r, even when I used the hint in the exercise. It simply never returned. As far as I can tell it is because the newline char used is \n, not \r\n. Instead I just wrote my own line reader.
* The hint in the exercise for reading from the switches and writing to the leds did not work – the IOWR\_ALTERA... and IORD\_ALTERA functions (or macro definitions) did not exist, but I just used direct assignment to PIO\_OUTPUT1\_BASE and reading from PIO\_INPUT\_BASE and that worked fine.
* I had to remember that the input and output is only 8 bit in the BSP and that is why I can only read 8 of the switches.
* There seems to be generated a new-line in the debugging console after the CMD:/>, and I am unsure where it comes from, but I believe it is the IDE, as I do not write it.
* I attempted to use fseek with the LCD display. Then I tried to figure out the table in the exercise, but finally I did an internet search and then got the correct VT100 commands. The shown spaces in the exercise hint are not allowed to be there.

I was a lot of fun and it worked perfectly. The code is included in the appendix.

# Assignment 4

I updated the design and started the timeout timer. Then I created a new project (Matrix\_mult) based on the BSP created in Assignment 2.

Instead of writing a command language interpreter I simply made a test run in main. This is due to the fact that everything is hard-coded – there is no need for input from the user.

I happened to write the code for the Soft implementation with two nested for-loops and a call to a multiply-accumulate function, which had a single for-loop. This fit perfect with the update to the Hard version, though I did not realise that until I started the next assignment.

Please refer to Assignment 5 for results and code.

# Assignment 5

I updated the design with the VDL Custom Instruction no problem, and updated the code (remembering to rebuild the BSP ☺).

I continued the previously created project implementing the Hard function and the calling code.

The result may be seen in the table below, and the code in the appendix.

|  |
| --- |
| AInst=  |1 2 3 4|  |5 6 7 8|  |9 10 11 12|  |13 14 15 16|  BInst=  |1 1 1 1|  |2 2 2 2|  |3 3 3 3|  |4 4 4 4|  PInst=  |10 20 30 40|  |26 52 78 104|  |42 84 126 168|  |58 116 174 232|  Calculation 4x4 Soft took: 11538 ticks  PInst=  |10 20 30 40|  |26 52 78 104|  |42 84 126 168|  |58 116 174 232|  Calculation 4x4 Hard took: 2303 ticks |

# Assignment 6

I updated the design as indicated in the exercise. Then I created a new project (Sweep\_generator) based on the BSP created in Assignment 2.

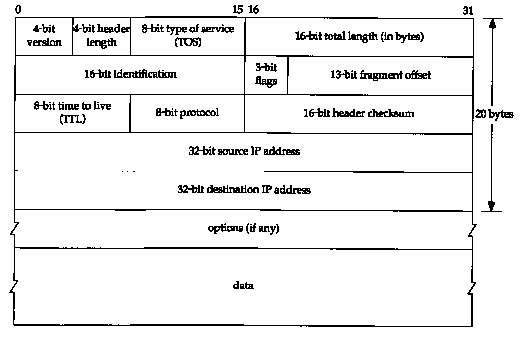
I updated the design with the WaveforGenerator and CodecInterface as indicated in the exercise, but unfortunately it did not work. I got a compilation error in Quartus II. After some digging I had to look at the hints, but finding nothing about this problem I wrote Kim. He quickly wrote back, and included a copy of his design for Assignment 6, so I could continue. I compared our two projects but was not able to find out what the problem. I sent my comparison and my entire project to Kim, who will look at it to see if he can find out what is wrong.

During the back and forth I was able to write most of the code, and there was some interesting challenges.

How to read the Registry diagrams. When I normally see a protocol description it will have a have the bit-pattern horizontally (like it does) and the layout of the content vertically (bit-fields), like e.g. the IP header specification (see Figur 1), and then having details about how to use the protocol somewhere else. That is not the case here. Here it seems to be only the top line that indicates the bit´-field layout, and the remaining lines showing direction, address and possible content (RESET). Actually the RESET line is very strange. First I figured: “This is what you have to write to the registry to reset the given registry, but that does not make much sense for the frequency or attenuation, as that would mean that attenuation = 1 is RESET and frequency 16 is reserved for RESET. So at first I tried to ignore the RESET and wrote the following:

|  |
| --- |
| IOWR(CODECINTERFACE\_0\_BASE,0x0000,0x0000); // RESET  // Write attenuation (not really necessary, see below)  IOWR(WAVEFORMGENERATOR\_0\_BASE, 0x00000000, attenuation);  while (i <= 255)  {  // Write frequency (and attenuation. Must be included every time as we write 32bit)  IOWR(WAVEFORMGENERATOR\_0\_BASE, 0x00000000, (i << 8) | attenuation);  usleep(delay);  ++i;  } |

As it may be seen the attenuation has to be written every time, and we only use Offset = 0. This is due to the fact that the offset mentioned in the registry diagrams are byte offset, and the offset mentioned in IOWR is DWORD offset – I discovered this while I was scrutinizing the registry diagrams; I had originally written it wrong. This is also where I realized that the attenuation has to be written every time.



Figur 1 - IP header description

With Kim’s project in hand I could build a BSP and use this to complete the assignment. When I ran the code (after some name updates) nothing happened. I chose the not very scientific approach and tried to change some values, like writing to the WAVEFORMGENERATOR\_0\_BASE + 1 (offset=1) and adding some delays (memory mapped control often need a little time to handle the value before you override it. This did not work.

Then I looked at the Codec Init registry and decided to try and set it to 1 (bitfield INIT = 1). This still not worked, and cheating a little and looking at the hints, I found out that one needed to set INIT = 1, wait a little and then RESET the Codec Init registry. This had the desired effect. See the code in the appendix for the final result.

Now whether this is the correct solution I have no idea, but sound is emitted and the tone is changing, but without knowing what it is suppose to sound like it is difficult, and without a description of how to use the different registries it is even more difficult.

I received an answer from Kim with the information that there are multiple instances of CPU\_System in my project design, as well as duplicate I/O pins. I( remember that when I choose Update Symbols and Block all the existing I/O pins disappeared so I had to re-create them. I did this by undoing the update, copying them into the clipboard, redoing the update and then pasting them in place. I must be here that something has gone wrong. Deleting the entire project and doing it from scratch with the WaveGenerator in place from the beginning would most likely work, but as Kim also wrote that I was allowed to use his design I have done so for this assignment, and the following one. Unfortunately for the first two assignments to work I have to have a design with features that is not included in Kim’s project , so for this reason I am going to try and create a design with all features enabled, as part of Assignment 8, and then move Assignment 6 and 7 to that BSP.

# Assignment 7

Following the Wizard to create the project was easy. There was information about where to find the semaphore, and the API was easy to guess, but then the simplicity stopped. I was unable to find any information about which header file contained the message queue and the mail box. I did some file searching, and find some references in system.h and other. I discovered a struct called os\_mbox\_data, and armed with this I decided to do a Google search, and found:

MicroC/OS-II Quick Reference: <http://sigpromu.org/brett/elec3730/quickref.html>

And that helped. I decided to use the suggestions for the semaphore from the Quick Reference instead of the ones from os/sem.h, that way the application is more symmetrical. I quickly finished the assignment and it worked perfectly.

There are many ways to solve this assignment, as the only requirements are a semaphore, a mail box and a message queue and then one consumer task and two producer tasks. Should both producer tasks write to the mail box and message queue, or can each task be dedicated to one type? Should the consumer task use the semaphore to event the producer tasks or the other way around? Here is what we know:

1. The mail box, semaphore and message queue causes an error is there is no more room, yet no simple way to query if there is room. This indicates that we should prevent them from overflowing (as opposed to handling the error, which is naturally also an option).
2. The mail box semaphore and message queue all have pending methods, used to block the thread until a new message or event is received. This is due to the fact that you normally have a dedicated consumer task which then simply blocks until something is available. This is not the case for us, but luckily the pend methods has a timeout, which we can simply set to 0 or ½ the total delay (we need to poll both channels). This results in a less than optimal solution, as there is a potential 1 the total delay before a message is received, and we cannot use delay = 0, without consuming 100% of the CPU (busy wait). The reason for this is that if we are waiting for a message on the mail box and someone sends one on the message queue, we have to wait for the mail box pend to complete before we can poll the message box. Using the pend timeout is still preferable to a sleep, as the the worst case delay is the total delay (not ½ the total delay).

I have chosen to use two semaphores, one for the mailbox and one for the mail box. The semaphores will contain the number of free slots in the message queue and mail box respectively, and will be used to block the producer tasks until they can safely write to the message queue or mail box.

I have chosen to let each producer task post to either the mail box or the message queue.

I have chosen to let the consumer task simply pend on first the mail box and then the message queue with a timeout of ½ the chosen delay. As the delay is an integer this means that the total delay might be reduced by 1ms.

I have also chosen to use relative timeouts for the producer tasks. This means that even though the producer task has been pending on a semaphore for a certain amount of time, it will still be subject to the full sleep. An alternative is to record how long the task has been pending on the semaphore and then reduce the sleep with that value. The gain is not considered worth the added complexity in this case.

# Assignment 8

# Conclusion

# Appendix A

|  |
| --- |
| **#include** <stdio.h>  **#include** <string.h>  **#include** <system.h>  **static** **void** **showInfo**()  {  **printf**("HW control interface\n");  **printf**("------------------------------\n");  **printf**("Protocol: [type] [target] [value] (ex. \"set led 128 \" | \"get sw \")\n");  **printf**("Types: get | set | exit\n");  **printf**("Targets: lcd1 | lcd2 | sw | led | ticks\n");  **printf**("Value: 0-255 (led) | 0-65535 (ticks) | Text (lcd)\n");  **printf**("CMD:/> ");  }  **static** **void** **handleSWCommand**()  {  **unsigned** **long** **int** switches;  // We have only set up an 8bit port!!!  **volatile** **unsigned** **char**\* pioBase = (**unsigned** **char**\*)PIO\_INPUT\_BASE;  switches = (**unsigned** **long** **int**)\*pioBase;  **printf**("Switches: 0x%lX\n", switches);  **printf**("CMD:/> ");  }  **static** FILE\* lcdHW = 0;  **static** **void** **handleLCDCommand**(**int** lcdLine, **const** **char**\* text)  {  **if** (lcdHW != 0)  {  **fseek**(lcdHW, 0, SEEK\_SET); // Always start at the beginning of the line  **switch** (lcdLine)  {  **case** 1:  **fwrite**("\x1B[1;1H", 1, 6, lcdHW);  **fwrite**("\x1B[K", 1, 3, lcdHW);  **fwrite**(text, 1, **strlen**(text), lcdHW);  **break**;  **case** 2:  **fwrite**("\x1B[2;1H", 1, 6, lcdHW);  **fwrite**("\x1B[K", 1, 3, lcdHW);  **fwrite**(text, 1, **strlen**(text), lcdHW);  **break**;  }  }  }  **static** **void** **handleLEDCommand**(**unsigned** **char** led)  {  **volatile** **unsigned** **char**\* pioBase = (**unsigned** **char**\*)PIO\_OUTPUT1\_BASE;  \*pioBase = led;  **printf**("CMD:/> ");  }  **typedef** **enum**  {  *CT\_GET*,  *CT\_SET*,  *CT\_EXIT*,  CT\_UNKNOWN  } CommandType;  **static** CommandType **getCommandType**(**const** **char**\* type)  {  **if** (**strcmp**(type, "get") == 0)  {  **return** *CT\_GET*;  }  **else** **if** (**strcmp**(type, "set") == 0)  {  **return** *CT\_SET*;  }  **else** **if** (**strcmp**(type, "exit") == 0)  {  **return** *CT\_EXIT*;  }  **return** *CT\_UNKNOWN*;  }  **typedef** **enum**  {  *TT\_SW*,  *TT\_LCD1*,  *TT\_LCD2*,  *TT\_LED*,  TT\_UNKNOWN  } CommandTarget;  **static** CommandTarget **getCommandTarget**(**const** **char**\* target)  {  **if** (**strcmp**(target, "led") == 0)  {  **return** *TT\_LED*;  }  **else** **if** (**strcmp**(target, "lcd1") == 0)  {  **return** *TT\_LCD1*;  }  **else** **if** (**strcmp**(target, "lcd2") == 0)  {  **return** *TT\_LCD2*;  }  **else** **if** (**strcmp**(target, "sw") == 0)  {  **return** *TT\_SW*;  }  **return** *TT\_UNKNOWN*;  }  **static** **void** **handleCmdLine**(**const** **char**\* cmdLine)  {  **char** type[64]; // Only this long for safety reasons  **char** target[64]; // Only this long for safety reasons  **int** integer;  // VERY VERY unsafe if the buffer is not sure to be big enough  **sscanf**(cmdLine, "%s%s", type, target);  **switch** (getCommandType(type))  {  **case** *CT\_GET*:  **switch** (getCommandTarget(target))  {  **case** *TT\_SW*:  handleSWCommand();  **break**;  **default**: // All others  **printf**("Invalid get target recieved: %s\n", target);  // **TODO**: Empty stdin buffer  showInfo();  **break**;  }  **break**;  **case** *CT\_SET*:  **switch** (getCommandTarget(target))  {  **case** *TT\_LCD1*:  handleLCDCommand(1, cmdLine + 3 + 1 + 4 + 1); // these integers should not be here.  **break**;  **case** *TT\_LCD2*:  handleLCDCommand(2, cmdLine + 3 + 1 + 4 + 1);  **break**;  **case** *TT\_LED*:  **sscanf**(cmdLine + 3 + 1 + 3 + 1, "%d", &integer);  **if** (integer < 0 || integer > 255)  {  **printf**("Invalid cmd set led recieved, out of range: %d\n", integer);  // **TODO**: Empty stdin buffer  showInfo();  }  handleLEDCommand(integer);  **break**;  **default**: // All others  **printf**("Invalid get target recieved: %s\n", target);  // **TODO**: Empty stdin buffer  showInfo();  **break**;  }  **break**;  **case** *CT\_EXIT*: // Simply ignore  **break**;  **case** *CT\_UNKNOWN*:  **printf**("Invalid cmd recieved: %s\n", type);  // **TODO**: Empty stdin buffer  showInfo();  **break**;  }  }  **void** **readCmdLine**(**char**\* cmdLine, **int** length)  {  **int** readBytes = 0;  **do**  {  **fread**(cmdLine + readBytes, 1, 1, stdin);  ++readBytes;  }  **while** (cmdLine[readBytes - 1] != '\n' && readBytes < length - 1);  cmdLine[readBytes - 1] = '\0'; // remove newline - had it been \r\n this would have to have been different  }  **int** **main**()  {  **char** cmdLine[64] = { '\0' };  // Open LCD driver  lcdHW = **fopen**(LCD\_NAME, "w");  **if** (lcdHW == 0)  {  **printf**("Failure initializing LCD, lcd commands will not work");  }  showInfo();  **do**  {  cmdLine[0] = '\0'; // Ensure that it is cleared  readCmdLine(cmdLine, **sizeof**(cmdLine));  handleCmdLine(cmdLine);  }  **while** (**strcmp**(cmdLine, "exit") != 0);  **if** (lcdHW != 0)  {  **fclose**(lcdHW);  }  **return** 0;  } |

Tabel - Assignment 3 code

|  |
| --- |
| **#include** "sys/alt\_timestamp.h"  **#include** "alt\_types.h"  **#include** "system.h"  **#include** <stdio.h>  **typedef** **union** {  **unsigned** **char** comp[4];  **unsigned** **int** vect;  } vectorType;  **typedef** vectorType VectorArray[4];  // Prototypes  **void** **setInputMatrices**(VectorArray A, VectorArray B);  **void** **displayMatrix**(VectorArray input);  **void** **multiMatrixSoft**(VectorArray A, VectorArray B, VectorArray P);  **void** **multiMatrixC2H**(VectorArray A, VectorArray B, VectorArray P);  **void** **multiMatrixHard**(VectorArray A, VectorArray B, VectorArray P);  **void** **setInputMatrices**(VectorArray A, VectorArray B)  {  A[0].comp[0] = 1;  A[0].comp[1] = 2;  A[0].comp[2] = 3;  A[0].comp[3] = 4;  A[1].comp[0] = 5;  A[1].comp[1] = 6;  A[1].comp[2] = 7;  A[1].comp[3] = 8;  A[2].comp[0] = 9;  A[2].comp[1] = 10;  A[2].comp[2] = 11;  A[2].comp[3] = 12;  A[3].comp[0] = 13;  A[3].comp[1] = 14;  A[3].comp[2] = 15;  A[3].comp[3] = 16;  B[0].comp[0] = 1;  B[0].comp[1] = 1;  B[0].comp[2] = 1;  B[0].comp[3] = 1;  B[1].comp[0] = 2;  B[1].comp[1] = 2;  B[1].comp[2] = 2;  B[1].comp[3] = 2;  B[2].comp[0] = 3;  B[2].comp[1] = 3;  B[2].comp[2] = 3;  B[2].comp[3] = 3;  B[3].comp[0] = 4;  B[3].comp[1] = 4;  B[3].comp[2] = 4;  B[3].comp[3] = 4;  }  **void** **displayMatrix**(VectorArray input)  {  **printf**("|%d %d %d %d|\n", input[0].comp[0], input[0].comp[1], input[0].comp[2], input[0].comp[3]);  **printf**("|%d %d %d %d|\n", input[1].comp[0], input[1].comp[1], input[1].comp[2], input[1].comp[3]);  **printf**("|%d %d %d %d|\n", input[2].comp[0], input[2].comp[1], input[2].comp[2], input[2].comp[3]);  **printf**("|%d %d %d %d|\n", input[3].comp[0], input[3].comp[1], input[3].comp[2], input[3].comp[3]);  }  **int** **dotProduct**(VectorArray A, VectorArray B, **int** row, **int** column)  {  **int** res = 0;  **int** j;  **for**( j = 0; j < 4; ++j)  {  res += A[row].comp[j] \* B[column].comp[j];  }  **return** res;  }  **void** **multiMatrixSoft**(VectorArray A, VectorArray B, VectorArray P)  {  **int** i = 0;  **int** j = 0;  **for**( i = 0; i < 4; ++i)  {  **for**( j = 0; j < 4; ++j)  {  P[i].comp[j] = dotProduct(A, B, i, j);  }  }  }  **void** **multiMatrixHard**(VectorArray A, VectorArray B, VectorArray P)  {  **int** i = 0;  **int** j = 0;  **for**( i = 0; i < 4; ++i)  {  **for**( j = 0; j < 4; ++j)  {  P[i].comp[j] = ALT\_CI\_MULTIPLIERADD\_INST(A[i].vect, B[j].vect);  }  }  }  // Global variables, therefore not static (file scope)  VectorArray aInst;  VectorArray bInst;  **int** **main**()  {  **int** ts;  VectorArray pInst;  **printf**("Setting AInst and BInst\n");  setInputMatrices(aInst, bInst);  **printf**("AInst=\n");  displayMatrix(aInst);  **printf**("BInst=\n");  displayMatrix(bInst);  ts = alt\_timestamp\_start();  multiMatrixSoft(aInst, bInst, pInst);  ts = alt\_timestamp() - ts;  **printf**("PInst=\n");  displayMatrix(pInst);  **printf**("Calculation 4x4 Soft took: %d ticks\n", ts);  ts = alt\_timestamp\_start();  multiMatrixHard(aInst, bInst, pInst);  ts = alt\_timestamp() - ts;  **printf**("PInst=\n");  displayMatrix(pInst);  **printf**("Calculation 4x4 Hard took: %d ticks\n", ts);  **return** 0;  } |

Tabel - Assignment 4 and 5 code

|  |
| --- |
| **#include** <stdio.h>  **#include** <string.h>  **#include** <stdlib.h> // usleep  **#include** "system.h"  **#include** <io.h> // IOWR  **static** **void** **showInfo**()  {  **printf**("Sweep generator\n");  **printf**("------------------------------\n");  **printf**("sweep <delay> <Attenuation>\n");  **printf**("delay (0-65535), Attnuation (0-15)\n");  **printf**("CMD:/> ");  }  **static** **void** **handleCmdLine**(**const** **char**\* cmdLine)  {  **int** delay;  **int** attenuation;  **int** i = 0;  **sscanf**(cmdLine, "sweep %d %d", &delay, &attenuation);  **if** (delay > 65535 || delay < 0)  {  **printf**("Delay out of range\n");  showInfo();  **return**;  }  **if** (attenuation > 15 || attenuation < 0)  {  **printf**("Attenuation out of range\n");  showInfo();  **return**;  }  IOWR(CODECINTERFACE\_0\_BASE,0x0000,0x0001); // INITIALIZE  usleep(100);  IOWR(CODECINTERFACE\_0\_BASE,0x0000,0x0000); // RESET  IOWR(WAVEFORMGENERATOR\_0\_BASE, 0x00000000, attenuation); // Write attenuation (not really necessary, see below)  **while** (i <= 255)  {  IOWR(WAVEFORMGENERATOR\_0\_BASE, 0x00000000, (i << 8) | attenuation); // Write frequency (and attenuation. Must be included every time as we write 32bit)  usleep(delay);  ++i;  }  IOWR(WAVEFORMGENERATOR\_0\_BASE, 0x00000000, 0x00); // Write frequency (and attenuation. Must be included every time as we write 32bit)  **printf**("Sweep done");  }  **void** **readCmdLine**(**char**\* cmdLine, **int** length)  {  **int** readBytes = 0;  **do**  {  **fread**(cmdLine + readBytes, 1, 1, stdin);  ++readBytes;  }  **while** (cmdLine[readBytes - 1] != '\n' && readBytes < length - 1);  cmdLine[readBytes - 1] = '\0'; // remove newline - had it been \r\n this would have to have been different  }  **int** **main**()  {  **char** cmdLine[64] = { '\0' };  showInfo();  **do**  {  cmdLine[0] = '\0'; // Ensure that it is cleared  readCmdLine(cmdLine, **sizeof**(cmdLine));  handleCmdLine(cmdLine);  }  **while** (**strcmp**(cmdLine, "exit") != 0);  **return** 0;  } |

Tabel - Assignment 6 code

|  |
| --- |
| **#include** <stdio.h>  **#include** "includes.h"  **#include** "system.h"  **#include** <os/alt\_sem.h>  /\* Definition of Task Stacks \*/  **#define** TASK\_STACKSIZE 2048  OS\_STK taskProducer1\_stk[TASK\_STACKSIZE];  OS\_STK taskProducer2\_stk[TASK\_STACKSIZE];  OS\_STK taskConsumer\_stk[TASK\_STACKSIZE];  /\* Definition of Task Priorities \*/  **#define** TASK\_PRODUCER1\_PRIORITY 1  **#define** TASK\_PRODUCER2\_PRIORITY 2  **#define** TASK\_CONSUMER\_PRIORITY 3  //ALT\_STATIC\_SEM(eventSemaphore)  **static** OS\_EVENT \* mailboxSemaphore;  **static** OS\_EVENT \* msgqueueSemaphore;  **static** OS\_EVENT\* mailBox;  **static** OS\_EVENT\* msgQueue;  **#define** MESSAGE\_QUEUE\_ENTRY\_COUNT 12  **static** **void**\* msgQueueBuffer[MESSAGE\_QUEUE\_ENTRY\_COUNT];  **static** **unsigned** **short** producer1TaskDelay;  **static** **unsigned** **short** producer2TaskDelay;  **static** **unsigned** **short** consumerTaskDelay;  **void** **taskMailboxProducer**(**void**\* pdata)  {  INT8U res;  **char** dataBuffer[64];  **int** i = 0;  **while** (1)  {  // timeout = 0 indicate indefinite  OSSemPend(mailboxSemaphore, 0, &res);  **if** (res == 0)  {  **sprintf**(dataBuffer, "Hello from Mailbox Process: %d", ++i);  **printf**("Adding message to mailbox\n");  res = OSMboxPost(mailBox, dataBuffer);  **if** (res != 0)  {  **printf**("MailboxProducer: Failure writing to mailbox: %hu\n", res);  }  }  **else**  {  **printf**("MailboxProducer: Failure getting semaphore: %hu\n", res);  }  OSTimeDly(producer1TaskDelay);  // OSTimeDlyHMSM(0, 0, 0, producer1TaskDelay);  }  }  **void** **taskMessageQueueProducer**(**void**\* pdata)  {  INT8U res;  **char** dataBuffer[MESSAGE\_QUEUE\_ENTRY\_COUNT][64];  **int** i = 0;  **int** messageIndex = 0;  **while** (1)  {  // timeout = 0 indicate indefinite  OSSemPend(msgqueueSemaphore, 0, &res);  **if** (res == 0)  {  **sprintf**(dataBuffer[messageIndex], "Hello from Message Queue Process: %d", ++i);  **printf**("Adding message to message queue\n");  res = OSQPost(msgQueue, dataBuffer[messageIndex]);  **if** (res != 0)  {  **printf**("MessageQueueProducer: Failure writing to Message Queue: %hu\n", res);  }  **if** (++messageIndex == MESSAGE\_QUEUE\_ENTRY\_COUNT)  {  messageIndex = 0;  }  }  **else**  {  **printf**("MessageQueueProducer: Failure getting semaphore: %hu\n", res);  }  OSTimeDly(producer2TaskDelay);  // OSTimeDlyHMSM(0, 0, 0, producer2TaskDelay);  }  }  **static** **void** **postToSemahore**(OS\_EVENT\* sem, **const** **char**\* semName)  {  **switch** (OSSemPost(sem))  {  **case** OS\_NO\_ERR:  **printf**("Posted to semaphore %s\n", semName);  **break**;  **case** OS\_SEM\_OVF:  **printf**("Failure posting to semaphore %s, overflow\n", semName);  **break**;  **case** OS\_ERR\_EVENT\_TYPE:  **printf**("Failure posting to semaphore %s, event type error\n", semName);  **break**;  }  }  **void** **taskConsumer**(**void**\* pdata)  {  INT8U errorRet;  // We could do this with just one variable  **void**\* mailboxResult;  **void**\* messagequeueResult;  **while** (1)  {  errorRet = 0;  // Pend on the mailbox  // ticks to ms (max tick = 65535)  mailboxResult = OSMboxPend(mailBox, consumerTaskDelay / 2, &errorRet);  **if** (mailboxResult != 0)  {  **printf**("Received message from mailbox: %s\n", (**const** **char**\*)mailboxResult);  // As the mailboxResult is a pointer, it is important that the  // pointer is not invalidated until we are done with it. We fix this by  // waiting until we are done with the pointer to release the mailbox.  // This means that the producer only need one pointer for data.  postToSemahore(mailboxSemaphore, "mbSem");  }  **else** **if** (errorRet == OS\_TIMEOUT) // Timeout  {  // Do nothing.  }  **else** // Error  {  **printf**("Error occured waiting on mail box: %d\n", errorRet);  }  errorRet = 0;  // Pend on the message queue  // ticks to ms (max tick = 65535)  messagequeueResult = OSQPend(msgQueue, consumerTaskDelay / 2, &errorRet);  **if** (messagequeueResult != 0)  {  **printf**("Received message from message queue: %s\n", (**const** **char**\*)messagequeueResult);  // As the messagequeueResult is a pointer, it is important that the  // pointer is not invalidated until we are done with it. We fix this by  // waiting until we are done with the pointer to release the message queue.  // This means that the producer "only" need 12 slots for data.  postToSemahore(msgqueueSemaphore, "msqSem");  }  **else** **if** (errorRet == OS\_TIMEOUT) // Timeout  {  // Do nothing.  }  **else** // Error  {  **printf**("Error occured waiting on message queue box: %d\n", errorRet);  }  // OSTimeDlyHMSM(0, 0, 0, consumerTaskDelay);  }  }  /\* The main function creates two task and starts multi-tasking \*/  **int** **main**(**void**)  {  // ALT\_SEM\_CREATE(eventSemaphore, 0);  mailboxSemaphore = OSSemCreate(1);  **if** (mailboxSemaphore == 0)  {  **printf**("Failure creating MailBox semaphore\n");  **return** -1;  }  msgqueueSemaphore = OSSemCreate(MESSAGE\_QUEUE\_ENTRY\_COUNT);  **if** (msgqueueSemaphore == 0)  {  **printf**("Failure creating MessageQueue semaphore\n");  **return** -1;  }  mailBox = OSMboxCreate(NULL);  **if** (mailBox == 0)  {  **printf**("Failure creating MailBox\n");  **return** -1;  }  msgQueue = OSQCreate(msgQueueBuffer, MESSAGE\_QUEUE\_ENTRY\_COUNT);  **if** (msgQueue == 0)  {  **printf**("Failure creating MessageQueue\n");  **return** -1;  }  **printf**("Please enter the Producer 1 Task run interval[ticks](0-65535): ");  **scanf**("%hu", &producer1TaskDelay);  **printf**("Please enter the Producer 2 Task run interval[ticks](0-65535): ");  **scanf**("%hu", &producer2TaskDelay);  **printf**("Please enter the Consumer Task run interval[ticks](0-65535): ");  **scanf**("%hu", &consumerTaskDelay);  OSTaskCreateExt(taskMailboxProducer,  NULL,  (**void** \*)&taskProducer1\_stk[TASK\_STACKSIZE-1],  TASK\_PRODUCER1\_PRIORITY,  TASK\_PRODUCER1\_PRIORITY,  taskProducer1\_stk,  TASK\_STACKSIZE,  NULL,  0);  OSTaskCreateExt(taskConsumer,  NULL,  (**void** \*)&taskConsumer\_stk[TASK\_STACKSIZE-1],  TASK\_CONSUMER\_PRIORITY,  TASK\_CONSUMER\_PRIORITY,  taskConsumer\_stk,  TASK\_STACKSIZE,  NULL,  0);  OSTaskCreateExt(taskMessageQueueProducer,  NULL,  (**void** \*)&taskProducer2\_stk[TASK\_STACKSIZE-1],  TASK\_PRODUCER2\_PRIORITY,  TASK\_PRODUCER2\_PRIORITY,  taskProducer2\_stk,  TASK\_STACKSIZE,  NULL,  0);  OSStart();  **return** 0;  } |

Tabel - Assignment 7 code

|  |
| --- |
|  |

Tabel - Assignment 8 code