Aarhus Universitet

ERTS - Group 2

Rapport

Assignment 2

Gruppemedlemmer:	Studienumre:
Daniel Tøttrup	201509520
Mathias Lønborg Friis	201505665
Stinus Lykke Skovgaard	201401682

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

4. oktober 2019

Kapitel 1

EXERCISE 3

Write a C-application that implements a command language interpreter, controlled via the USB-UART interface. The following commands must be implemented:

- **1 - Sets the binary value from 0-15 on the red led's by reading switch input (SW0-SW3)**
- **2 - Counts binary the red led's using a timer of 1 sec.**

1.1 Command 1

The essence of the first command is to set a binary value from 0-15 by using the four switches, which will be represented as a binary value by the four LED's on the board.

The code snippet below shows how we read input from the terminal, and execute the a command based on that input.

```c
//read input from the terminal in byte
int userInput = inbyte();
xil_printf("Received: %d\r\n", userInput);

switch(userInput)
{
case (int)'1':
xil_printf("Received command 1\r\n");
execute_command_1();
break;
case (int)'2':
xil_printf("Received command 2\r\n");
execute_command_2();
break;
case (int)'3':
xil_printf("Received command 3\r\n");
execute_command_3();
break;
case (int)'4':
xil_printf("Received command 4\r\n");
execute_command_4();
```

```
break;
default:
xil_printf("Received unknown command.");
break;
}
```

Below is a code snippet of the implemented first command.
When the function **execute_command_1()** is called the switches on the board is read with the function **XGpio_DiscreteRead(&dip, 1)**. Where &dip is the InstancePtr that points to the XGpio instance that is being worked on, and the '1' is the channel.

The **writeValueToLEDs(int val)** takes the value returned from the **XGpio_DiscreteRead()** and writes the value to the LED registers.

```
void execute_command_1()
{
xil_printf("Executing command 1\r\n");
while(1)
{
dip_check = XGpio_DiscreteRead(&dip, 1);

writeValueToLEDs(dip_check);

}
}

void writeValueToLEDs(int val)
{
LED_IP_mWriteReg(XPAR_LED_IP_S_AXI_BASEADDR, 0, val);
}
```

Command 1 work as expected.

1.2 Command 2

Command 2 should, as states in the problem description, count the binary red LED's using the timer of 1 sec.

First the timer needs to be initialized. Below is a code snippet of how the timer is implemented. Comments in the code explains the implementation.

```
#define ONE_SECOND 325000000 // half of the CPU clock speed

// PS Timer related definitions
XScuTimer_Config *ConfigPtr;
```

```c
XScuTimer *TimerInstancePtr = &Timer;

int main (void)
{
//initialize timer
ConfigPtr = XScuTimer_LookupConfig (XPAR_PS7_SCUTIMER_0_DEVICE_ID);
s32 Status = XScuTimer_CfgInitialize (TimerInstancePtr, ConfigPtr,
    ConfigPtr->BaseAddr);
if (Status != XST_SUCCESS){
xil_printf("Timer init() failed\r\n");
return XST_FAILURE;
}

// Load timer with delay in multiple of ONE_SECOND
XScuTimer_LoadTimer(TimerInstancePtr, ONE_SECOND);
// Set AutoLoad mode
XScuTimer_EnableAutoReload(TimerInstancePtr);
}
```

The function **execute_command_2()** sets op a counter and writes the value of counter to the LED's with the function **writeValueToLEDs(counter)** which was explained above. Then it starts the timer, and waits for the timer to expire, which will take one second. When the timer has expired the timer is cleared and the counter is incremented, before the loop starts again. Below is a code snippet of command 2.

```c
void execute_command_2()
{
xil_printf("Executing command 2\r\n");
int counter = 0;
while(1)
{
xil_printf("Counter: %d\r\n", counter);
writeValueToLEDs(counter);

// Start the timer
XScuTimer_Start(TimerInstancePtr);

// Wait until timer expires
while(!XScuTimer_IsExpired(TimerInstancePtr));

// clear status bit
XScuTimer_ClearInterruptStatus(TimerInstancePtr);

counter++;

// Timer auto-enables

}
```

The function **execute_command_2()** works as expected.

Kapitel 2

EXERCISE 4

This exercise will deal with the steps to implement a matrix multiplier.

At first three global variables of the data structure **vectorArray** called **pInst**, **aInst** and **bTInst** were created.

Next a function called **setInputMatrices()** was implemented. This function will initialize two matrices. Below is a code snippet of the implementation. The comments in the code explains how the each element in the matrices are initialized.

```
void setInputMatrices()
{
//Set matrix a
// A, row 0
aInst[0].comp[0] = 1;
aInst[0].comp[1] = 2;
aInst[0].comp[2] = 3;
aInst[0].comp[3] = 4;

// A, row 1
aInst[1].comp[0] = 5;
aInst[1].comp[1] = 6;
aInst[1].comp[2] = 7;
aInst[1].comp[3] = 8;

// A, row 2
aInst[2].comp[0] = 9;
aInst[2].comp[1] = 10;
aInst[2].comp[2] = 11;
aInst[2].comp[3] = 12;

// A, row 3
aInst[3].comp[0] = 13;
aInst[3].comp[1] = 14;
aInst[3].comp[2] = 15;
aInst[3].comp[3] = 16;

//Set matrix bT
// A, row 0
bTInst[0].comp[0] = 1;
bTInst[0].comp[1] = 2;
```

```
bTInst[0].comp[2] = 3;
bTInst[0].comp[3] = 4;

//Set matrix bT
// A, row 1
bTInst[1].comp[0] = 1;
bTInst[1].comp[1] = 2;
bTInst[1].comp[2] = 3;
bTInst[1].comp[3] = 4;

//Set matrix bT
// A, row 2
bTInst[2].comp[0] = 1;
bTInst[2].comp[1] = 2;
bTInst[2].comp[2] = 3;
bTInst[2].comp[3] = 4;

//Set matrix bT
// A, row 3
bTInst[3].comp[0] = 1;
bTInst[3].comp[1] = 2;
bTInst[3].comp[2] = 3;
bTInst[3].comp[3] = 4;
}
```

Next a function called **displayMatrix(vectorArray matrix)** was implemented. As the name of the function strongly implies, this function will display a given matrix in the terminal. To implement **displayMatrix(vectorArray matrix)** a function called **displayMatrixRow(vectorArray matrix, int row)** is implemented in addition. That's because we have to display the matrix row for row. Below is a code snippet of the implemented function.

```
void displayMatrix(vectorArray matrix)
{
xil_printf("[\r\n");
displayMatrixRow(matrix,0);
displayMatrixRow(matrix,1);
displayMatrixRow(matrix,2);
displayMatrixRow(matrix,3);
xil_printf("]\r\n");
}

void displayMatrixRow(vectorArray matrix, int row)
{
xil_printf("%d %d %d %d \r\n", matrix[row].comp[0],
    matrix[row].comp[1], matrix[row].comp[2], matrix[row].comp[3]);
}
```

Next a function called **multiMatrixSoft(vectorArray in1, vectorArray in2, vectorArray out)** was implemented, which is responsible for computing 4x4 matrix product of the expression

$$P = AxB^T$$

This function takes three input because its not possible to return a matrix. So the result matrix is given as the third input parameter, that way we can set the new value of the result matrix in the function. To iterate all combinations of the result matrix P to calculate the sum of products of each element in P, three nested for-loops were used. Below is a code snippet of the **multiMatrixSoft(vectorArray in1, vectorArray in2, vectorArray out)**

```
void multiMatrixSoft(vectorArray in1, vectorArray in2, vectorArray out)
{
  for (int row = 0 ; row < MSIZE; row++)
  {
    for (int col = 0 ; col < MSIZE; col++)
    {
      xil_printf("[%d,%d]\r\n", row, col);
      for (int i = 0 ; i < MSIZE ; i++)
      {
        out[row].comp[col] += in1[row].comp[i] * in2[row].comp[i];
      }
    }
  }
};
```

Next, we wanted to test the multiMatrixSoft function, and test the execution time in clock ticks. For this we implemented a timer, and two integers: **time_SW_pre** and **time_SW_post**. We use the **XScuTimer_GetCounterValue(TimerInstancePtr)** to get the elapsed time before and after the execution and store it in the variables **time_SW_pre** and **time_SW_post**. We then subtract **time_SW_post** form **time_SW_pre** to get the excution time in clock ticks. Below is the a code snippet of this with comments.

```
 XScuTimer Timer; /* Cortex A9 SCU Private Timer Instance */

 // PS Timer related definitions
 XScuTimer_Config *ConfigPtr;
 XScuTimer *TimerInstancePtr = &Timer;

// Multiply matrices in SW.
xil_printf("Calculating P in SW\r\n");
int time_SW_pre, time_SW_post;
XScuTimer_Start(TimerInstancePtr);  // Start the timer
time_SW_pre = XScuTimer_GetCounterValue(TimerInstancePtr); // Get elapsed time
```

```
multiMatrixSoft(aInst, bTInst,pInst);
time_SW_post = XScuTimer_GetCounterValue(TimerInstancePtr); // Get elapsed time
int time_elapsed_SW = time_SW_pre-time_SW_post; // Calculate elapsed time
xil_printf("P = \r\n");
displayMatrix(pInst);

xil_printf("SW: %d\r\n",timeElapsed_SW); //print execution time
```

```
Serial: (COM5, 115200, 8, 1, None, None - CONNECTED) - Encoding: (ISO-8859-1)
-- Start of the Program --

Commands:
1 - Set binary value of LED by reading switch input.
2 - Timer counts using red LEDs.
3 - Matrix multiplication3.
Received: 51
Received command 3
Executing command 3
Matrix multiplication
A =
[
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
]
bT =
[
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
]
Calculating P in SW
[0,0]
[0,1]
[0,2]
[0,3]
[1,0]
[1,1]
[1,2]
[1,3]
[2,0]
[2,1]
[2,2]
[2,3]
[3,0]
[3,1]
[3,2]
[3,3]
P =
[
30 30 30 30
70 70 70 70
110 110 110 110
150 150 150 150
]

Multiplication times (clock cycles):
SW: 0
```

Figur 2.1: multiMatrixSoft result in terminal

Kapitel 3

EXERCISE 5

Kapitel 4

EXERCISE 7