COMP8049 – Embedded Systems Engineering
Lab 2

Completion Date: 4th of December 2020

Value: 15 marks

Question 1

The attached game-lab directory which contains a basic “game” code for the versatilepb board. Modify the code to allow the user to move pacman around the versatilepb board screen without leaving a trail. The user can use keys such as s,d,e and w to move pacman. You need to use the arm cross compiler and the mk script.

Question 2

Give a brief overview of the clang generated Abstract Syntax Tree (AST) for the while loop of the attached t2.c file. The attached t2.ast file from line 25 to line 53 shows a dump of the WhileStmt node and its children which corresponds to the while loop. The t2.ast file was generated using:

```
clang-check -ast-dump t2.c --extra-arg="-fno-color-diagnostics" >t2.ast
```

Clang/LLVM is a framework for the compilation and analysis of source code written is several different programming languages. Clang provides many tools for the manipulation of its ASTs. See the video linked in this tutorial for details https://jonasdevlieghere.com/understanding-the-clang-ast/.

In this case we use clang to generate an AST for a given simple c function fred(). We use the -ast-dump command line argument to generate a textual representation of the AST, which is stored in the file t2.ast. Each node in the AST models a particular language construct. For example, the VarDecl class models a c declaration of a variable.

For instance, the c statement in the file t1.c:

```
int i,j=0;
```

is modeled in the AST as two VarDecl nodes and one IntegerLiteral node which are in the t2.ast from line 15 to line 18:

```
-VarDecl 0x1c08820 <col:1, col:5> col:5 used i 'int'
-VarDecl 0x1c08890 <col:1, col:9> col:7 used j 'int' cinit
   -IntegerLiteral 0x1c088f0 <col:9> 'int' 0
```

If a c variable is used in an expression in t2.c, the AST will have a corresponding DeclRefExpr node to model that use.

For example, the c statement:

j = 0;

will be modeled using the following AST nodes (dumped using the -ast-dump flag to the compiler):

```
BinaryOperator 0x1c089f8 <line:4:1, col:3> 'int' '='
   | |-DeclRefExpr 0x1c089b0 <col:1> 'int' lvalue Var 0x1c08890 'j' 'int'
   | `-IntegerLiteral 0x1c089d8 <col:3> 'int' 0
```

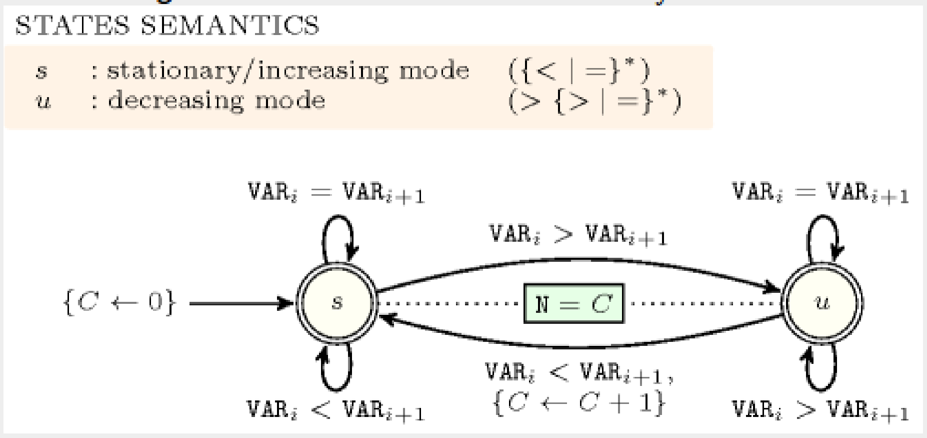
Question 3

The attached code (testbench.c and peak.c) is a testbench and a state machine implementation of the peak constraint http://sofdem.github.io/gccat/gccat/Cpeak.html. You are to write a c function which implements the “valley” constraint and a second c function which implements the “increasing peak” constraint. You are to update the testbench to test these new constraints.

A constraint is just a form of if statement. The *valley* and *increasing peak* constraints are defined here: http://sofdem.github.io/gccat/gccat/Cvalley.html http://sofdem.github.io/gccat/gccat/Cincreasing_peak.html.

The *valley* constraint is a simple state machine, which is practically identical to the *peak* constraint. An implementation of the peak constraint state machine is given in peak.c. See figure 5.417.3 below for the valley constraint state machine.

Figure 5.417.3. Automaton of the valley constraint


STATES SEMANTICS
s : stationary/increasing mode ($\{<|=\}^*$)
u : decreasing mode ($> \{>|=\}^*$)
$\{C \leftarrow 0\}$
s
u
$\mathtt{VAR}_i = \mathtt{VAR}_{i+1}$
$\mathtt{VAR}_i = \mathtt{VAR}_{i+1}$
$\mathtt{VAR}_i > \mathtt{VAR}_{i+1}$
$\mathtt{N} = C$
$\mathtt{VAR}_i < \mathtt{VAR}_{i+1}$
$\mathtt{VAR}_i < \mathtt{VAR}_{i+1}$, $\{C \leftarrow C+1\}$
$\mathtt{VAR}_i > \mathtt{VAR}_{i+1}$

The scope of this lab is the design of a hardware accelerator for detecting anomalies in time series data. These anomalies are defined by the user in terms of constraints which can be used to detect combinations of spikes and/or troughs with particular characteristics. Such accelerators could be deployed for example into a nuclear power facility to quickly detect fluctuations in the sensor readings from the instrumentation control systems of the plant. http://www.mcobject.com/radico, https://www.iaea.org/NuclearPower/IandC/index.html

The different constraints can be combined to form more complex filters. See for example here
https://www.doulos.com/knowhow/systemc/tutorial/modules_and_processes/

where a systemc model of an EXOR gate is implemented using four NAND gates.

The code (testbench.c and peak.c) is a high level data flow model/design that can be synthesised directly onto a FPGA using for example vivado HLS.
https://www.xilinx.com/video/hardware/getting-started-vivado-high-level-synthesis.html