Software Design

Friday 27 March 2015

2IO70

The purpose of this document is to present a Java program that realises the functions specified in the Software Specification document. This program is an intermediate step towards writing the PP2 code that controls the sorting machine.

Group 16

Rolf Verschuuren

Wigger Boelens

Stefan van den Berg

Dat Phung

Maarten Keet

Tudor Petrescu

Version 2

Table of Contents

Coding Standards

The java pseudo code follows the Google Java Style.
Source to Google Java Style: https://google-styleguide.googlecode.com/svn/trunk/javaguide.html.
PHP code used in this project follows the Zend Framework Coding Standard for PHP.
Source: http://framework.zend.com/manual/1.12/en/coding-standard.html.

Translating to pseudo java:

The java program starts by declaring the output variables. The names of the output variables will keep their original name, without spaces, in a camelCase form. The variable type will be determined from the Output table.
The inputs follow the same pattern.

Every states is represented as a function, keeping their name in the camelCase fashion, they will be all void functions due to the fact that they do not return anything.
Every state function will run preconditions if any, then check for specific input values using if statements, if an if statement is satisfied, there will be changes to the output values to match the next states output values, also the display is set to output the next states number, and then the next state function is called according to the state transition diagram, if no if statement is satisfied the current function is recalled.

The program is always looping, consequence of no deadlocks in the state machine as proven by the uppaal model test.

Example: Initial -> Calibrate_Sensor
So in this example the function initial is currently running, there are no preconditions to be checked, if the inputs have the desired value, in our case we check if the push button is pressed by the sorter, if so we will have the sorter moved down by activating the sorter motor via having the Hbridge0 variable set to 1. After this we set the display to showcase the number $branchTO where to branch to2 then call calibrateSensor function and if the if statement wasn't satisfied we recall initial entering a loop.

Translating from Java to PHP

The java code was written such that the conversion process to php is as easy as possible.
All variable in java will have the "$" sign added at the beginning of their name to comply with the php standards. The "$" sign has no influence in the java program variable naming, while in php it is mandatory.

Design decisions for the Java code

In translating our transition table to a Java program we made a number of decisions shaping the code, these decisions are outlined in this section.

We started by looking at our transition table, in this table we had our transitions ordered by the "current state", the state where the transition starts. Then there were some inputs that could trigger a transition from this state to a number of other states. Because of this we thought it would make sense to write a function for each state, since it would allow our code to essentially be a condensed version of the transition table. Where the code would be ordered by the "current state", and each state would have a number of outgoing transitions to other states. This resulted in the following blueprint for each of our functions:

```
void function(){
  timerManage();              // The function that manages the outputs and PWM
  $temp = getButtons();       // Store the buttons currently being pressed in a temporary variable

  if( condition for transition ){       // Do this once for every outgoing transition from this state
    Changes required to change to the new statement;
  }

  function();                 // Call on the same function again to recheck the buttons and
continue
                                    running the machine with timerManage()
}
```

Then we made an extra function which will be called from each function to do the PWM. This function is called timerManage. This function firstly gets the voltage which the output needs from the array.
This function has a variable called counter which increments each time the outputs have been set. That value is take modulo 12. So it will leave the outputs which need 12 volts on all the time. The reason why the values which need less than 12 volt will be turned off after they have been on for long enough. That goes as follows. First it checks if the engine needs to be on by checking if the voltage it needs is higher than counter. If the output needs to be on then it gets the location of the value in the array. And then does 2 to the power of the location. So now the correct output will be set on. Then the value of 2 to the power will be added to the variable engines. Then after all 7 outputs have been through that loop then it will set the output to the value of engines. So the lights which needed to be on will be on. Now the value of counter will increment each time and take modulo 12.

We also choose to save certain values, which may not be expected to be saved. In this section I will explain why we save the 2 variables. The first one is the variable of the location of the code. This has been saved because then we then we are capable of changing the return address after the timer interrupt. Because when an timer interrupt occurs we want to return to the initial state and the position where we were before. We also saved the original position of the stack pointer for when we come back from the timer interrupt to make sure that we empty the stack. Because there may be some values on the stack from before the timer interrupt. Thus to remove them we set the stack pointer to its original value.

Validation

Validation of java to transition table

Every state is represented by a function. The if statements in that function are the transitions which can occur from that state. The timer interrupt and the abort transitions are not represented as if statements, because interrupts go to a separate state(function). In those if statements the values that have to change are changed. The display will also be updated to the correct number of the state. The function timerManage is called in each state. Because with that function we make sure that the all outputs have the correct voltage.
We checked that all states are represented in the java code by a function. We also checked if they have all the transitions as if statements and that the correct values are changed.

Validation of timerManage

Loop invariant:
All elements before the current element of the array have been set on if they had to be on.

Initialize:
We start with the first element. Thus there are no elements before it and the loop invariant holds.

Step case:
If we're at element k, then according to the loop invariant all elements before k have been set on if they had to be on. Then if k has to be on (value of k>counter) it will be set on else it will stay off. So now the loop invariant holds for the element k+1

Termination:
The loop will terminate when k is greater than 7. Because we do not have any more outputs.

Control flow validation

Because the Java code has been validated to the state description and the transition table, which, in turn, have been validated with the UPPAAL model and shown to be correct and in tune with the initial description of the sorting machine. This means that the Java program, being a one-to-one translation of the finite state automaton, also has a correct control flow.

Appendix: Java Program

```
/**
 * Sort of a simulation of the PP2 program
 * controlling the Fischer
 * Technik in order to sort black and white discs.
 *
 * @author Maarten Keet
 * @author Stefan van den Berg
 * @author Rolf Verschuuren
 * @author Wigger Boelens
 * @team Group 16
 * @since 13/3/2015
 */


class SoftwareDesign {
    //**@CODE**
    //inputs
    int $push, $startStop, $abort, $position,
            $colour;

    //variables
    int $state = 0;
    int $sleep = 0;
    int $temp = 0;
    int $location;
    int $counter = 0;
    int $engines;


    //constants
    final int TIMEMOTORDOWN = 30;
    final int BELTROUND = 2000;
    final int BELT = 1200;
    final int SORT = 850;
    final int LENSLAMPPOSITION = 5,
            LENSLAMPSORTER = 6,
            HBRIDGE0 = 0,
            HBRIDGE1 = 1,
            CONVEYORBELT = 3,
            FEEDERENGINE = 7,
            DISPLAY = 8,
            LEDSTATEINDICATOR = 9;

    public static void main(String args[]) {
        SoftwareDesign SoftwareDesign = new
                SoftwareDesign();


        //values for the data segment
        SoftwareDesign.initVar("outputs", 12);
        SoftwareDesign.initVar("stackpointer", 1);
        SoftwareDesign.initVar("offset", 1);

        //store the offset of the programm,this
        // is used in the interrupt
        SoftwareDesign.storeData(startofthecode,
                                 "offset", 0);

        //store the vlue of the stackpointer,so
        // we can clear the stack
        // easily
        SoftwareDesign.storeData(SP,
                                 "stackpointer",
                                 0);


        $counter = 0;


        //reset outputs
        SoftwareDesign.storeData(0, "outputs",
                                 SoftwareDesign
                                .HBRIDGE1);
        SoftwareDesign.storeData(0, "outputs",
                                 SoftwareDesign
                                .LENSLAMPPOSITION);
        SoftwareDesign.storeData(0, "outputs",
                                 SoftwareDesign
                                .LENSLAMPSORTER);
        SoftwareDesign.storeData(0, "outputs",
                                 SoftwareDesign
                                .LEDSTATEINDICATOR);
        SoftwareDesign.storeData(0, "outputs",
                                 SoftwareDesign
                                .DISPLAY);
        SoftwareDesign.storeData(0, "outputs",
                                 SoftwareDesign
                                .CONVEYORBELT);
        SoftwareDesign.storeData(0, "outputs",
                                 SoftwareDesign
                                .FEEDERENGINE);

        //start moving the sorter up
        SoftwareDesign.storeData(9, "outputs",
                                 SoftwareDesign
                                 .HBRIDGE0);


        //go to the first state and set the
        // value for the display
        SoftwareDesign.$state = 0;
        SoftwareDesign.initial();
    }

    //state0
    void initial() {
        setStackPointer(
                getData("stackpointer", 0));
        timerManage();
        //check if the sorter push button is
        // pressed
        $push = getButtonPressed(5);
        if ($push == 1) {
            //move the sorter down
            storeData(0, "outputs", HBRIDGE0);
            storeData(9, "outputs", HBRIDGE1);
            //update the state
            $state = 1;
            //reset sleep for the next function
            $sleep = 0;
            calibrateSorter();

        }
        //loop
        initial();
    }

    //state 1
    void calibrateSorter() {
        timerManage();
        //the sorter is now moving down,
        //and we're waitng for it to reach the
        // bottom
        if ($sleep == TIMEMOTORDOWN * 1000) {
            //stop the sorter
            storeData(0, "outputs", HBRIDGE1);
            //update the state
            $state = 2;
            //reset sleep
            $sleep = 0;
            resting();
        }
        //loop
        $sleep++;
        calibrateSorter();
    }

    //state 2
    void resting() {
        timerManage();
        //the program waits for the user to
        // press the start/stop
        $startStop = getButtonPressed(0);
        if ($startStop == 1) {
            //sleep so we don't go to the pause
            // immediatly
            sleep(2000);
            //power up the lights
            storeData(12, "outputs",
                      LENSLAMPPOSITION);
            storeData(12, "outputs",
                      LENSLAMPSORTER);
            //start up the belt and the feeder
            storeData(9, "outputs", CONVEYORBELT);
            storeData(5, "outputs", FEEDERENGINE);
            //set and start the countdown
            setCountdown(BELTROUND + BELT);
            startCountdown();
            //update the state
            $state = 3;
            running();
        }
        //loop
        resting();
    }

    //state 3
    void running() {
        timerManage();
        //check if we need to pause
        $startStop = getButtonPressed(0);
        if ($startStop == 1) {
            //stop the feeder engine
            storeData(0, "outputs", FEEDERENGINE);
            //set the timer
            setCountdown(BELT);
            //update the state
            $state = 9;
            runningTimer();
        }
        //check if a disk is at the position
```

```
        // detector
        $position = getButtonPressed(7);
        if ($position == 1) {
            //reset the countdown,because a
            // disk was detected
            setCountdown(BELTROUND + BELT);
            //update the state
            $state = 4;
            runningWait();
        }
        //loop
        running();
    }

    void runningWait() {
        timerManage();
        //check if we need to pause
        $startStop = getButtonPressed(0);
        if ($startStop == 1) {
            //stop the feeder engine
            storeData(0, "outputs", FEEDERENGINE);
            //set the timer
            setCountdown(BELT);
            //update the state
            $state = 9;
            runningTimer();
        }
        //check if a disk is at the positiond
        // detector
        $position = getButtonPressed(7);
        if ($position == 1) {
            //reset the countdown,because a
            // disk was detected
            setCountdown(BELTROUND + BELT);
            //update the state
            $state = 5;
            runningTimerReset();
        }
        //check if a white disk is at the color
        // detector
        $colour = getButtonPressed(6);
        if ($colour == 1) {
            //move the sorter up
            storeData(9, "outputs", HBRIDGE0);
            //update the state
            $state = 6;
            motorUp();
        }
        //loop
        runningWait();
    }

    //state 5
    void runningTimerReset() {
        timerManage();
        //update the state
        $state = 5;
        runningWait();
    }

    //state 6
    void motorUp() {
        timerManage();
        //check if we need to pause
        $startStop = getButtonPressed(0);
        if ($startStop == 1) {
            //stop the feeder engine
            storeData(0, "outputs", FEEDERENGINE);
            //set the timer
            setCountdown(BELT);
            motorUpTimer();
        }
        //check if the sorter push button is
        // pressed
        $push = getButtonPressed(5);
        if ($push == 1) {
            //stop the engine,because it is in
            // the right position
            storeData(0, "outputs", HBRIDGE0);
            //update the state
            $state = 7;
            whiteWait();
        }
        //loop
        motorUp();
    }

    //state 7
    void whiteWait() {
        timerManage();
        //we are waiting for the white disk to
        // be sorted
        if ($sleep == SORT * 1000) {
            //start moving the sorter down
            storeData(9, "outputs", HBRIDGE1);
            //update the state
            $state = 8;
            //reset sleep for the next function
            $sleep = 0;
            motorDown();
        }
        //check if we need to pause
        $startStop = getButtonPressed(0);
        if ($startStop == 1) {
            //stop the feeder engine
            storeData(0, "outputs", FEEDERENGINE);
            //set the timer
            setCountdown(BELT);
            //update the state
            $state = 11;
            whiteWaitTimer();
        }
        //loop
        $sleep++;
        whiteWait();
    }

    //state 8
    void motorDown() {
        timerManage();
        //the sorter is moving down
        if ($sleep == TIMEMOTORDOWN * 1000) {
            //stop the sorter
            storeData(0, "outputs", HBRIDGE1);
            //update the state
            $state = 9;
            //reset sleep for the next function
            $sleep = 0;
            runningWait();
        }
        //check if we need to pause
        $startStop = getButtonPressed(0);
        if ($startStop == 1) {
            //stop the feeder engine
            storeData(0, "outputs", FEEDERENGINE);
            //set the timer
            setCountdown(BELT);
            motorDownTimer();
        }
        //loop
        $sleep++;
        motorDown();

    }

    //state 9
    void runningTimer() {
        timerManage();
        //update state
        $state = 13;
        runningStop();
    }

    //state 10
    void motorUpTimer() {
        timerManage();
        //update state
        $state = 14;
        motorUpStop();
    }

    //state 11
    void whiteWaitTimer() {
        timerManage();
        //update state
        $state = 15;
        whiteWaitStop();
    }

    //state 12
    void motorDownTimer() {
        timerManage();
        //update state
        $state = 16;
        motorDownStop();
    }

    //state 13
    void runningStop() {
        timerManage();
        //check if a white disk is at the
        // colour detector
        $colour = getButtonPressed(6);
        if ($colour == 1) {
            //move the sorter engine up
            storeData(9, "outputs", HBRIDGE0);
            //update the state
            $state = 10;
            motorUpStop();
        }
        //loop
        runningStop();
    }

    //state 14
    void motorUpStop() {
        timerManage();
        //check if the sorter push button is
        // pressed
```

```
        $push = getButtonPressed(5);
        if ($push == 1) {
            //stop the engien for the sorter
            storeData(0, "outputs", HBRIDGE0);
            //update the state
            $state = 11;
            whiteWaitStop();
        }
        motorUpStop();
    }

    //state 15
    void whiteWaitStop() {
        timerManage();
        //check if the white disk has been sorted
        if ($sleep == SORT * 1000) {
            //start moving the sorter down
            storeData(9, "outputs", HBRIDGE1);
            //update the state
            $state = 12;
            //reset the sleep for the next
            // function
            $sleep = 0;
            motorDown();
        }
        //loop
        $sleep++;
        whiteWaitStop();
    }

    //state 16
    void motorDownStop() {
        timerManage();
        //check if the sorter has moved down
        if ($sleep == TIMEMOTORDOWN) {
            //stop the engine of the sorter
            storeData(0, "outputs", HBRIDGE1);
            //update the state
            $state = 9;
            //reset sleep for the next function
            $sleep = 0;
            runningWait();
        }
        //loop
        $sleep++;
        motorDownStop();
    }

    //not a state
    void timerInterrupt() {
        //show that we have timer interrupt
        $state = 18;
        //make the sorter move up
        storeData(9, "outputs", HBRIDGE0);
        //stop all other outputs
        storeData(0, "outputs", HBRIDGE1);
        storeData(0, "outputs", LENSLAMPPOSITION);
        storeData(0, "outputs", LENSLAMPSORTER);
        storeData(0, "outputs",
                  LEDSTATEINDICATOR);
        storeData(0, "outputs", DISPLAY);
        storeData(0, "outputs", CONVEYORBELT);
        storeData(0, "outputs", FEEDERENGINE);
        //make sure that the outputs get set
        // immediatly
        timerManage();
        //set the display to the state of initial
        $state = 0;

        initial();

    }

    void abort() {
        //stop all outputs
        storeData(0, "outputs", HBRIDGE0);
        storeData(0, "outputs", HBRIDGE1);
        storeData(0, "outputs", LENSLAMPPOSITION);
        storeData(0, "outputs", LENSLAMPSORTER);
        storeData(0, "outputs",
                  LEDSTATEINDICATOR);
        storeData(0, "outputs", DISPLAY);
        storeData(0, "outputs", CONVEYORBELT);
        storeData(0, "outputs", FEEDERENGINE);
        //make sure the outputs stop immediatly
        timerManage();
        //update the state to be correct in
        // aborted
        $state = 17;
        aborted();

    }

    //state 17
    void aborted() {
        timerManage();
        //check if we can start again
        $startStop = getButtonPressed(0);
        if ($startStop == 1) {
            //start moving the sorter up for
            // calibration
            storeData(1, "outputs", HBRIDGE0);
            //update the state
            $state = 0;
            initial();
        }
        //loop
        aborted();

    }

    void timerManage() {


        //make sure that when counter can not
        // be higher than 12
        mod(13, $counter);
        //get the voltage of output $location
        int $voltage = getData("outputs",
                               $location);
        //power up the output when it needs to
        if ($voltage > $counter) {
            $engines += pow(2, $voltage);
        }
        //check if we are in a new itteration
        if ($counter == 0) {
            //set the first part of the display
            $temp = getData("state", 0);
            mod(10, $temp);
            display($temp, "display", "1");


        }
        //check if we are at the end of the
        // itteration
        if ($counter == 12) {
            //set the second part of the display;
            $temp = getData("state", 0);
            $temp = $temp / 10;
            mod(10, $temp);
            display($temp, "display", "01");

        }
        //check if we did all outputs
        if ($location > 7) {
            display($engines, "leds", "");
            //set the variables for the next run
            $engines = 0;
            $location = 0;
            $counter++;

            //check if abort is pressed
            $abort = getButtonPressed(1);
            if ($abort == 1) {
                abort();//stop the machine
            }
            return;
        }


        $location++;
        timerManage();
    }
}
```