Exercise 5.2c

12-2-2015

2IO70

Group 16

Stefan

Wigger

Version 1.0

Contents

Introduction 3
Program Interface 3
Test cases 3
Program structure 5
Appendices 7
Appendix 1 7
Appendix 2 12
Appendix 3 14

Introduction

The machine should be able to regulate the intensity of the LEDs on the PP2 Board. The intensity of the lights will be changed using a technique called PWM. With this technique you can change the intensity of a light by having it switching it on and off really quickly and regulating the amount of time that it's on. The light switches so fast that our eyes won't see it flickering. The processor can easily beat the time that it takes for our eyes to see the flickering which is 10 milliseconds. To use the PWM technique we let the light on only for a certain part of those 10 milliseconds then the light will be less intense than when it is on all the time. For example when the light blinks 8 out of the 10 times in those 10 milliseconds then the intensity will be just 80%.

Program Interface

The interaction with the machine is via the buttons and the potmeter on the pp2-processor. If one of the buttons from 1 to 7 is pressed, the intensity of the corresponding light should increase by 10%. If one of those buttons is pressed while button 0 is pressed, then it should decrease the intensity of the light by 10%. The intensity of the LED corresponding to button 0 is being controlled by the potmeter.

Test cases

The first test case is to check if the program interface is good enough for the requirements of the machine. Is it possible to change the intensity of the lights with the program interface?

This is possible because when a button from 1 to 7 is pressed the intensity of a light increases. When button 0 is pressed at the same time the intensity decreases. Thus we can change the intensity and the requirements are met.

To provide full decision coverage we will need the following test cases. We will need an input of button 1 and also an input of button 1 and button 0 to have full decision coverage.

The expected output when button 1 is pressed is that LED 1 will increase in intensity. The expected output when button 1 and button 0 is pressed is that LED 1 will decrease in intensity.

When we tested both test cases on the PP2-board and in both cases the outcome was as expected.

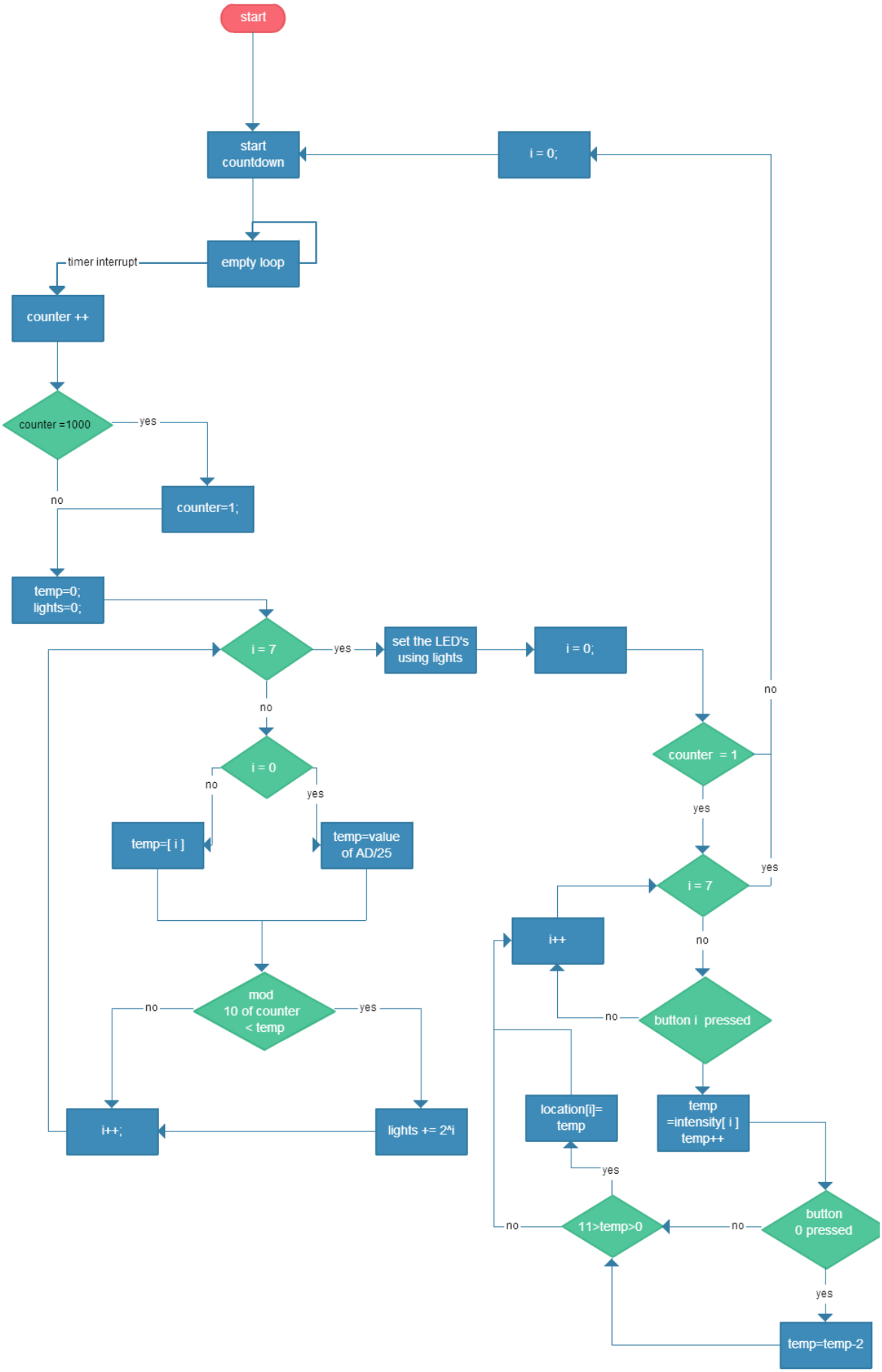

figure 1: Control flow

Program structure

The program is written in php(appendix 1). The php code will then be run through a compiler which is described in appendix 2. The assembly code can be found in appendix 3.

The structure of the php code of the program consist of a couple of functions. But before the functions are called a couple of variables are saved in the memory.

First an array called “intensity” with length 8 ,for each light one, and a variable called “counter” is created. Then the first function is called which is the main function(figure 2 lines 1-3). In this function the timer interrupt is initialized, and all values in the registers get reset to 0 and counter in the memory gets set to 0.The timer interrupt is initialized by telling the microcontroller to which point in the program it needs to branch when a timer interrupt occurs. Next the program branches to the init function.

In init(figure 2 lines 4-5) all values in the “intensity” array get set to 0. It will also start the timer and set it to 1.This is set to 1 because then it will take hundred microseconds. Why it needs to be a hundred microseconds will be explained later
Then it calls the function emptyloop(figure 2 line 7). Which as the name suggest does nothing except for looping towards itself. Then when the timer gets to 0 a timer interrupt occurs and the function loop will be called.
In loop(figure 2 lines 9-15) the timer gets once again set to 1. In loop the variable counter will be incremented by 1. If counter becomes equal to 1000, then it will be set back to 1. We wait until it is a 1000, because then 1 millisecond has passed. We need it to be 1 millisecond because then we can change the intensity by 10% (10 times 1 millisecond= 10 milliseconds) . Since the human eye doesn’t see flickering when it is faster than 10 milliseconds we see the change in intensity. The program then continues with the function getValues.
In getValues(figure 2 lines 16-23) a number will be stored in the variable lights. That value represents which lights should be on. This value is calculated by checking at what intensity each light has to burn. If that value is higher than the modulo 10 of the counter, then the number which represents that LED will be added to lights. With that value all the lights which need to be on are activated. Then the program then continues to checkButtons.
In checkButtons there will be checked for input only once per 1000 interrupts. This is done only once per 1000 interrupts, because else the intensity of the light would increase too fast and you can’t control it good enough.

As you can see in figure 2 (lines 24 -33) the intensity of the light will only decrease if button i is pressed, the counter is equal to 1 and button 0 is pressed. It will increase when a button I is pressed and the counter is equal to 1. In any other case it will return from the exception and go back to the emptyloop.

```
1  Initialise the timer
2  Set intensity, counter, location to 0
3  Store counter in the memory
4  For i=0 to i=7
5  Store intensity in the memory on location i
6  Start the countdown
7  Loop for ever
8  Timer interrupt:
9  Reset the countdown to 1
10 Load the value of counter from the memory
11 counter++
12 If counter=1000
13 counter=1
14 Store the value of counter in the memory
15 Set temp, lights to 0
16 For i=0 to i=7
17 if location=0
18       store the value of the AD in temp
19       divide temp by 25
20 else
21       get the value of the memory on location i and save it in temp
22 if the mod 10 of counter < temp
23       lights+=2^i
24 if counter=1
25 For i = 0 to i = 7
26       if button i is pressed
27             get the value of the memory on location i and save it in temp
28             temp++
29             if button 0 is pressed
30                   temp = temp - 2
31             if 11 > temp > 0
32                   save the value of temp in the memory on location i
33 go back to the loop on line #6
```

figure 2

Appendices

Appendix 1

```
<?php
include 'functions.php';
//everything above //**DATA** is left out
//**DATA**
initVar('intensity', 8);
initVar('counter', 1);

//**CODE**
define('WAIT', 1);//define a value for wait

function main()
{
    global $intensity, $location, $counter, $lights;//just for php purposes  doesn't reoccur in the assembly program
    installCountdown('loop');//initializes the timer interrupt
    $intensity = 0;//is stored in R0
    $counter = 0;//is stored in R1
    $location = 0;//is stored in R2

    _storeData($counter, 'counter', 0);//stores counter in the global base

    init();//calls the init function
}

function init()
{
    global $location, $intensity;//just for php purposes  doesn't reoccur in the assembly program
```

```
    $location++;//increments $location
    _storeData($intensity, 'intensity', $location);//stores $intensity at $location
    if ($location == 7) {//checks if it went through all the locations
        setTimer(WAIT);//set the timer at value wait
        startCountdown();//starts the timer
        emptyLoop();//calls the function emptyloop
    }
    init();//calls itself
}

function emptyLoop()
{
    emptyLoop();//keeps calling itself
}

function loop()
{
    global $counter, $location, $lights, $temp;//just for php purposes  doesn't reoccur in the
assembly program

    setTimer(WAIT);//set the timer at value wait
    startCountdown();//starts the countdown



    $counter = _getData('counter', 0);//get the value of $counter
    $counter++;//increments $counter

    if ($counter == 1000) {//checks if timer has run 10 times since when it was set to 1
        $counter = 1;//make the $counter ready for the next call
    }
```

```
    _storeData($counter, 'counter', 0);//stores $counter in the GB



    $location = -1;//set the $location to the AD
    $lights = 0;//initialize the value of $lights
    $temp = 0;//initialize the value of $temp
    getValues();//call the function getValues
}


function getValues()
{
    global $counter, $location, $lights, $temp;//just for php purposes  doesn't reoccur in the
assembly program
    //get all values to see which lights should be on or off


    $location++;//increment $location
    if ($location == 0) {//check if the $location is the location of the AD
        getInput($temp, 'analog');//set $temp to the value of the AD
        $temp /= 25;//divide by 25 to make it between 0 and 10


    }
    if ($location != 0) {//get the value of the light
        $temp = _getData('intensity', $location);//set $temp to intensity of that light
    }


    modulo($counter,10);


    if ($counter < $temp) {//check if the light should be on


        stackPush($lights);//put $lights on the stack
        pow(2, $location);//get the right $location of the LED result gets stored in R5
```

```
        stackPull($lights);//pull $lights of the stack
        $lights += R5;//creates the value of which LEDs need to be on
    }

    $counter = _getData('counter', 0);//because we used mod earlier, we need to get $counter
again

    if ($location == 7) {//check if each value of all lights have been added to $lights
        display($lights, 'leds');//set the LEDs on according to the value $lights
        $location = 0;//set $location back to 0
        checkButtons();//call checkbuttons
    }
    getValues();//call itself
}

function checkButtons()
{
    global $counter, $location, $temp, $lights;//just for php purposes  doesn't reoccur in the
assembly program
    if ($counter != 1) {
        //otherwise he checks the buttons too often, resulting in it going straight to full ON
        returnt;//returns from the timer-interrupt
    }
    $location++;//increment $location
    //check button 1
    buttonPressed($location);//set a 1 or a 0 depending on the state of the button at the $location
in R5
    if (R5 == 1) {
        $temp = _getData('intensity', $location);
        $temp++;//increment $temp

        buttonPressed(0);//check if button 0 is pressed value gets put in R5
```

```
        if (R5 == 1) {//button 0 is pressed
            $temp -= 2;//decrement $temp by 2
        }
        //make sure that the intensity of the light is between 0 and 10
        if ($temp != 11) {
            if ($temp != -1) {
                _storeData($temp, 'intensity', $location);
            }
        }
    }
    //
    if ($location != 7) {//check if all buttons have been checked
        checkButtons();//if not then call itself
    }
    returnt;//returns from the timer the timer-interrupt
}
```

Appendix 2

The compiler works in phases. We will go through these phases 1 by 1 to explain how the compiler does its job: compiling PHP-like code to assembly. Throughout the phases the compiler keeps track of the line number of the PHP code it is currently compiling and uses that, when an error occurs, to give information where the error is. The compiler is written in PHP5.6 and uses a command line interface.

Preprocessing

In the first phase, the input code will be made ready for the next steps. A few things happen in this phase: First the file is read into the memory. The next step is that all comments, newlines and extra spaces are stripped from the file. The file is then split into single lines using the ";" symbol that denotes the end of a line. While doing this the compiler writes the data to two arrays: the data array for everything between "//**DATA**" and "//**CODE**" and the code array for everything after "//**CODE**". Everything before //**DATA** is ignored. The data array gets compiled immediately.

The preprocessor further removes some special statements that are needed to make valid php such as "global" and changes some shortcuts in their full version. For example $abc++ will be changed into $abc+=1. This ensures that the compiler only needs to be able to handle $abc+=1.

Splitting

In the second phase the code is split up by function. Every function gets his own array with all the lines that are in that function. The code not inside of a function goes into a separate array.

Compiling

The third phase is the most important one. It starts by compiling the code that is at the start and not inside a function. While compiling it keeps track of what functions are called and adds these, if they are not already compiled, to the toCompile queue. This helps in making sure there is no dead code, as a function that is never called, will not be compiled. The compiler adds the function "main", which is the default start point of the code, to the queue and starts processing it.

After compiling the main function it will continue in the next function in the toCompile queue and keep doing this till the toCompile queue is empty.

The compiling itself is not a lot more than a lot of regex and switch statements that look at the input and make a output from that. At the first notion of a variable a register is assigned to it. The code then uses this register in place of the variable. Some more difficult statements, like the function display which displays something, will BRS to premade assembly code that handles that. The compiler keeps track of which segments of the premade assembly code are used.

When the compiler meets an if statement, it saves the code inside it to a new function named "condtionali" where i is the amount of conditionals that have already been seen. It then places this function in the toCompile queue. It also saves the location of the end of the if statement, so it will later know where to return when the if function has ended.

Combining

After there are no functions left in the toCompile queue, the combining phase starts. In this

phase all the functions and the code outside the functions are combined into a single array. This phase also adds the used premade functions at the top and inserts the return statements at the correct position.

Formatting

The last phase is the last interesting. It goes through the, now compiled code, and formats it. It uses either the length of the longest function name or the number 25 depending on which is larger to insert spaces in front of every line of code in a way everything lines up nicely.

The last step the compiler takes is writing the compiled code to a file and using the assembler provided to create the hex code.

Appendix 3

```
@DATA
intensity DS 8
counter DS 1

@CODE

                WAIT EQU 1
begin:          BRA main


                ;pow
_pow:           CMP R4 0
                BEQ _pow1
                CMP R4 1
                BEQ _powR
                PUSH R3
                PUSH R4
                SUB R4 1
                LOAD R3 R5
_powLoop:       MULS R5 R3
                SUB R4 1
                CMP R4 0
                BEQ _powReturn
                BRA _powLoop
_powReturn:     PULL R4
                PULL R3
                RTS
_pow1:          LOAD R5 1
                RTS
_powR:          RTS

                ;pressed
_pressed:       PUSH R4
                LOAD R4 R3
                LOAD R5 2
                BRS _pow
                LOAD R3 R5
                LOAD R5 -16
                LOAD R4 [R5+7]
                DIV R4 R3
                MOD R4 2
                LOAD R5 R4
                PULL R4
                RTS
```

```
45  main:               ;Install timer
46                      LOAD R0 loop
47                      ADD R0 R5
48                      LOAD R1 16
49                      STOR R0 [R1]
50
51                      LOAD R5 -16
52
53                      ; Set the timer to 0
54                      LOAD R0 0
55                      SUB R0 [R5+13]
56                      STOR R0 [R5+13]
57                      LOAD R0 0
58                      LOAD R1 0
59                      LOAD R2 0
60                      STOR R1 [GB +counter + 0]
61                      BRA init
62                      BRA main
63
64  init:               ADD R2 1
65                      ADD R2 intensity
66                      STOR R0 [ GB + R2]
67                      SUB R2 intensity
68                      CMP R2 7
69                      BEQ conditional0
70  return0:            BRA init
71                      BRA main
72
73  conditional0:       LOAD R5 -16
74                      LOAD R4 0
75                      SUB R4 [R5+13]
76                      STOR R4 [R5+13]
77                      LOAD R4 WAIT
78                      STOR R4 [R5+13]
79                      SETI 8
80                      BRA emptyLoop
81                      BRA return0
82                      BRA main
83
84  emptyLoop:          BRA emptyLoop
85                      BRA main
```

```
86
87  loop:               LOAD R5 -16
88                      LOAD R4 0
89                      SUB R4 [R5+13]
90                      STOR R4 [R5+13]
91                      LOAD R4 WAIT
92                      STOR R4 [R5+13]
93                      SETI 8
94                      LOAD R1 [ GB + counter + 0 ]
95                      ADD R1 1
96                      CMP R1 1000
97                      BEQ conditional1
98  return1:            STOR R1 [GB +counter + 0]
99                      LOAD R2 -1
100                     LOAD R3 0
101                     LOAD R4 0
102                     BRA getValues
103                     RTE
104
105 conditional1:       LOAD R1 1
106                     BRA return1
107                     BRA main
108
109 getValues:          ADD R2 1
110                     CMP R2 0
111                     BEQ conditional2
112 return2:            CMP R2 0
113                     BNE conditional3
114 return3:            MULS R4 100
115                     CMP R1 R4
116                     BMI conditional4
117 return4:            CMP R2 7
118                     BEQ conditional5
119 return5:            BRA getValues
120                     BRA main
121
122 conditional2:       LOAD R5 -16
123                     LOAD R4 [R5 + 6]
124                     DIV R4 25
125                     BRA return2
126                     BRA main
127
128 conditional3:       ADD R2 intensity
129                     LOAD R4 [ GB + R2]
130                     SUB R2 intensity
131                     BRA return3
132                     BRA main
```

```
133
134 conditional4:      PUSH R3
135                   LOAD R4 R2
136                   LOAD R5 2
137                   BRS _pow
138                   PULL R3
139                   ADD R3 R5
140                   BRA return4
141                   BRA main
142
143 conditional5:      LOAD R5 -16
144                   LOAD R4 R3
145                   STOR R4 [R5+11]
146                   LOAD R2 0
147                   BRA checkButtons
148                   BRA return5
149                   BRA main
150
151 checkButtons:      CMP R1 1
152                   BNE conditional6
153 return6:          ADD R2 1
154                   PUSH R3
155                   LOAD R3 R2
156                   BRS _pressed
157                   PULL R3
158                   CMP R5 1
159                   BEQ conditional7
160 return7:          CMP R2 7
161                   BNE conditional11
162 return11:         RTE
163                   BRA main
164
165 conditional6:     RTE
166                   BRA return6
167                   BRA main
168
169 conditional7:     ADD R2 intensity
170                   LOAD R4 [ GB + R2]
171                   SUB R2 intensity
172                   ADD R4 1
173                   PUSH R3
174                   LOAD R3 0
175                   BRS _pressed
176                   PULL R3
177                   CMP R5 1
178                   BEQ conditional8
```

```
179 return8:            CMP R4 11
180                     BNE conditional9
181 return9:            BRA return7
182                     BRA main
183
184 conditional8:       SUB R4 2
185                     BRA return8
186                     BRA main
187
188 conditional9:       CMP R4 -1
189                     BNE conditional10
190 return10:           BRA return9
191                     BRA main
192
193 conditional10:      ADD R2 intensity
194                     STOR R4 [ GB + R2]
195                     SUB R2 intensity
196                     BRA return10
197                     BRA main
198
199 conditional11:      BRA checkButtons
200                     BRA return11
201                     BRA main
202
203                     @END
```