**Modern Computer Architecture**

Date: 10/12/2023

**Important:**

1. Turn on **tracked changes** on **Microsoft Word document and use for the documentation of your work from the beginning to the end of your assignment.**
2. **You are to submit your work in Microsoft word format for your final submission.**
3. **You are to submit two versions of your work (please follow the right submission links)**
   1. **One with tracked changes still turned on**
   2. **One with all changes accepted and showing no markups**

**‘*The module tutor reserves the right to invite students to a meeting to discuss coursework submissions’***

* ***Providing links to external sources does not justify copy and paste of code.***
* ***Write your own code.***
* ***Do not start the assignment by remixing Tinkercad/other designs written by others.***

**Deliverables:**

1. Links to your designs in Tinkercad. ***Take a snapshot of the last edited information, as well as the image of your final design and paste below each link.***
2. Links of demos recorded or hosted on **Kaltura (make all links visible on the front page)**. **Please make all links visible on the front page and ensure that the videos can be watched with the links you provide**. Note that this is one way of rectifying that the final Submission is your own work, **has high grade weighting** and hence **you must submit in order to get a pass grade.**
   1. In your recording, give a detailed explanation as to how your code works
   2. Remember to use different tools such as ***whiteboard or Paint, datasheets, Arduino IDE for memory analysis*** etc. to explain underlining principles into details.
   3. Wherever possible use live coding to demonstrate your strengths
   4. If you started off your code with Arduino functions, such as “Pinmode()”, and have managed to use direct register manipulation to improve your code, remember to demonstrate the logic behind it in both your report and the video.
3. **Flowcharts (and algorithms, snapshots of your ideas).** Note, this is one way of showing how you have developed your solution
4. Program code with comments - this must be in **TEXT** format. No screenshots allowed, except of codes which I have provided you and have given you permission to use as part of your implementation (in which case, there will be no credit assigned to such code).
5. Evaluation of the output – Provide your observation, reflection and analysis on how you have optimized your code and design.

**Important:**

* **Your code must be in text format**.
* Use “**Kaltura**” for all your videos and make sure that the module tutor can view the video and **Tinkercad** with the link you provide. Please ensure that you links work.

Assignment

Using an 8-bit microcontroller based on the AVR enhanced RISC architecture (**ATmega328P**) in TinkerCad Environment, complete the following:

**For Part 1 & 2, Provide two solutions:**

1. **Requirement 1: Solution 1 with Arduino Macros: e.g. pinMode(), Delay, digitalWrite(), etc.**
2. **Requirement 2: Solution 2 with Register Manipulation: e.g. DDR etc. and register equivalents of interrupts etc.**
3. **Requirement 3: Show impact of different solutions on memory usage/space and how you have managed to improve the efficiency of your code.** 
   1. **Work with final solution that achieve all the functionalities with minimum memory requirements in the optimized code will achieve higher grade reward.**
4. **Requirement 4: Show the impact of Solution 1 and Solution 2 on SRAM and use it to explain the various sections of the SRAM and how it relates to your codes.**

**Part 1: Energy Efficient Decoration Lights**

Graphical user interface

Description automatically generated

Source:https://www.google.com/url?sa=i&url=https%3A%2F%2Fgoodwinelectrical.com%2F2018%2F12%2F16%2Fchristmas-lights-energy-efficiency%2F&psig=AOvVaw3bo1U7QVsqwDDja5qkQvZP&ust=1639770283569000&source=images&cd=vfe&ved=0CAsQjRxqFwoTCICCqsCK6fQCFQAAAAAdAAAAABAE

**Task**

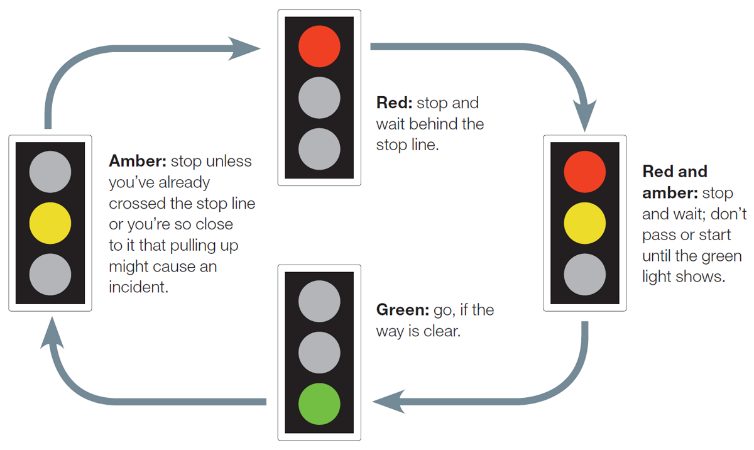
1. Create a prototype energy efficient Decoration/Christmas/Festive light.
2. Use a combination of **several RGBs** and a **few LEDs** for the lights
3. The light should only come on during the dark and automatically go off during the daytime. **The sensitivity threshold for daylight should be adjustable by the user.**
4. When the daylight intensity reduces to the threshold, the light should come on for **only 5-7 seconds** to show that it is activated.
   1. Then it should turn on **only when motion** is detected within its range with a default maximum **on time of 20 seconds** (**this should be adjustable** (an input device is required) **by the user**). **The time should reset only when the maximum on time has elapsed.**

**Note:**

* + 1. You will need a motion sensor and potentiometers
    2. **The processor should be configured to stop whatever it is doing to respond to a sensed motion.**
    3. When a motion is sensed, allow at least **two seconds** to pass before the processor responds to newly sensed motion: make use of the **Millis()** function.
  1. Add a feature that allows the user to **set the minimum distance an obstacle should be from the light** before the motion detection is used to activate the light. This distance should be adjustable (an input device is required) by the user with an input. **Note:**
     1. You will need another sensor that is different from a motion sensor.
  2. Add an option for **manual mode where the light can stay on without timing** out.

1. Use a remote control (use **input pushbuttons not IR remote** **control**) to change the lighting effect. This should allow the output color of the RGB LED to be adjusted with pre-set colors via short-cut color buttons (including white) and also, with one button to cycle through the colors: jump, stroboscopic, gradual and smooth color change. In your report, **provide your definition for these features:** jump, stroboscopic, gradual and smooth color change:
   * 1. The processor should be configured to stop whatever it is doing to respond to the pushbutton. **The logic level that the processor responds to with this button should be different from the logic level it responds to the motion sensor.**
     2. make use of the **Milli()** function wherever possible
2. **Provide dimming control** features.

**Part 2**: **Hello World (Traffic Light)**



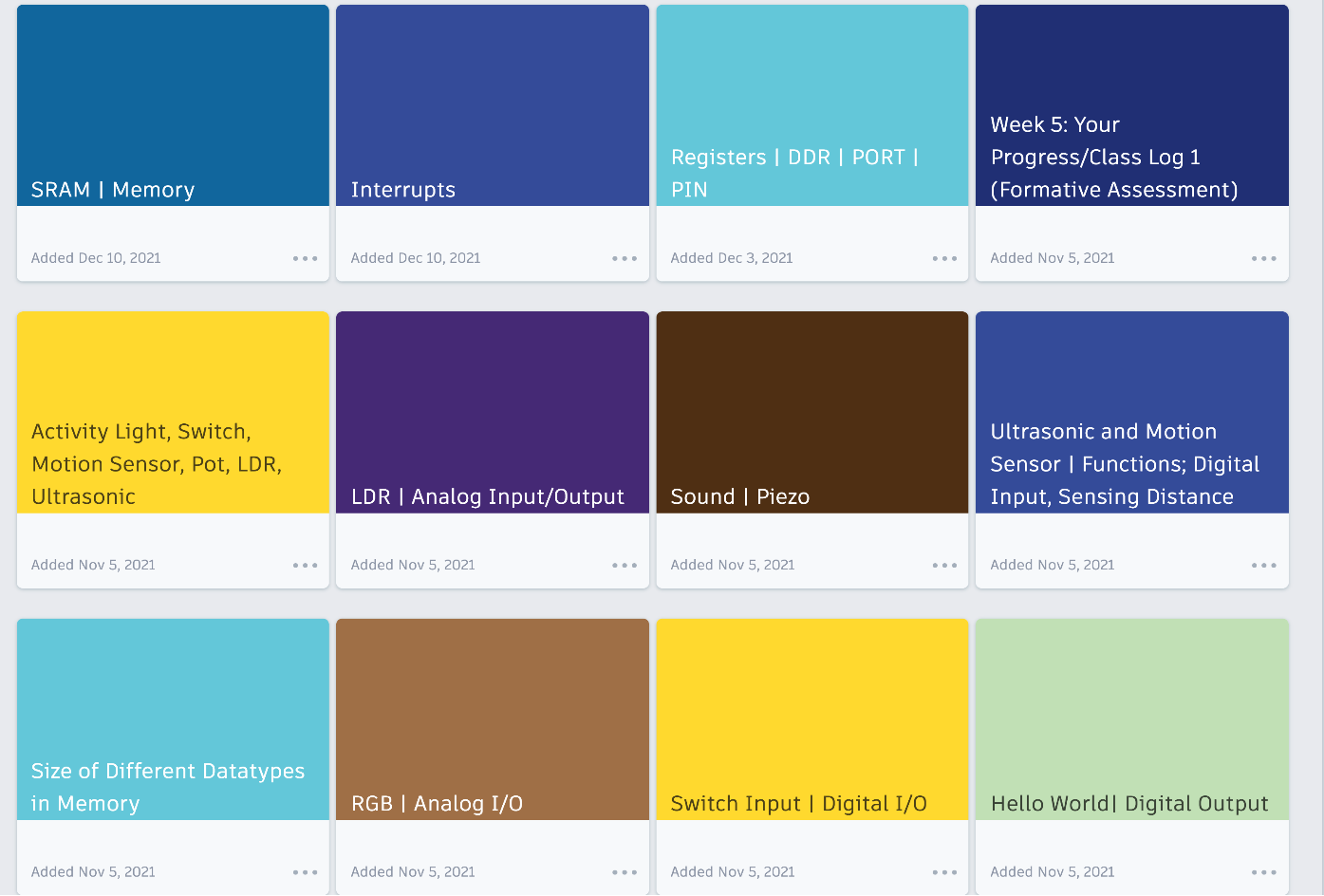
1. Design a traffic light that follows the above sequence.
2. When a pedestrian presses a pushbutton, and there is no motion detected on the road, it should quickly go to the red state after a few microseconds
   1. The processor should be configured to stop whatever it is doing to respond to the pushbutton.
   2. You should give the pedestrian enough time to cross the road.
   3. Use the Millis() function wherever possible
3. If the push button has not been pressed and the time for pedestrian to cross the road has lapsed and motion is detected on the road, quickly cycle to the green state.
4. Add sound to give audible status to pedestrian with visual impairment
5. Put a vibration motor on the floor to use different cues for pedestrians with visual impairment
6. Use an RGB on the pedestrian side to show the status of the LED lights on the roadside
7. Use LCD display to show the status of the lights:
   1. Use multiple LCDs (make use of animated Characters wherever possible)
      1. 1 or 2 for status of lights – including time remaining before next state
      2. 1 for advertisements

**Part 3 (Worth 10%)**

**You must complete Part 1 and 2 before attempting this part. Else, this part will attract 0 credits**

**Task**

1. For each of the following activities, submit your creative and most advanced version.



1. Highlight what you have done differently from the original activity and why and justify its impact on the output
2. Note: it is expected that each student will have a different implementation as this task is subject to your own creativity.
3. You are **NOT** required to submit flow diagrams or algorithms for this part
4. You are expected to provide your code in text and a snapshot of the final solution
   1. Sections of code that have directly been copied from **my solution** should be **provided as snapshots**. This should not exceed 60% of your solution
   2. **Your own code** should be provided in text.
5. Recommendation: provide a brief video recording to demonstrate each output and explain what you have done differently and its effect on the output

**Make sure you read the section on “How to Document Programs”!**

Dr. Michael Opoku Agyeman

**Hand in Date: 23/01/2022**

**How to Document Programs**

1. The two golden rules when writing programs are:
   1. Whenever possible, test sections of code you want individually. Don’t write the complete program and then try to find faults.
   2. **Do some design!** Think the program through before you begin writing your code.
2. There are two general types of people who will read your programs, namely:

* *The user of the program* and *someone who may want to modify it*.

The **user** of a program needs to know **how to run** it, what it needs and **what it does**. Thus, the beginning of your program should have the following information:

* A general statement about what the program does. Notice this is **what** it does, not **how** it does it.
* How to run the program, including any information on options and arguments.
* Any active program requirements. This includes any input the user must provide.
* What output the program generates and where it goes?
* Any assumptions the program makes about its environment. For example, if your program expects to be able to write to the current directory, you should say so. If it needs to use standard data from a particular place this must be documented, including where it must be, and its format.

The goal of documentation in the program header is to get the user to read it! This means that too much documentation is as bad as insufficient documentation. Avoid documenting things that are obvious. Keep documentation concise and organize it so that it is easy to read, with the most important points at the beginning or otherwise highlighted.

Example Header:

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Produced By: Computer Systems Engineering @ UoN

-- URL: <https://mypad.northampton.ac.uk/smartsc/>

-- Author: Prof. Michael Opoku Agyeman

-- Date: Yesterday/This Month/This Year

-- Purpose: To do something cool!

-- Modifications: Mr. somebody else \*today\* fixed my bug -- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

Each **function** and interrupt routine should also have a header e.g.

/\*--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-

-- Date: Yesterday/This Month/This Year

-- Purpose: To do something cool!

-- Called by: some other Entity/process

-- Modifications: Mr. somebody else \*today\* fixed my bug --\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* --\*/

The second audience is a person who may **wish to modify it**. This person will be an embedded system ‘C’ programmer with expertise as extensive as yours. This person needs to know how your program does what it does. This type of documentation needs to be in the body of the code, with the pieces of code it documents. Remember, this person has expertise, so the only things necessary to document are things that are not completely obvious. If you are using an assignment statement, don’t explain how it works if it is obvious. A comment explains what an output may do in the real world will be more helpful.

Documentation in the body of the code (called *inline documentation*) should not be added to the right of the code. Instead add entire lines of the documentation above the section of the code being documented.

Although it should be obvious, ensure that your documentation is correct.

Documentation is so important it will be worth 15% of the grade of each program.

**Style**

Your program should be easy to read. This means that the flow of control should be obvious, the names of the variables descriptive and the design as simple as possible.

Make use of white space. Use it for logical blocks of code that are processed similarly:

* put a blank line after a closing } that ends a block.
* put spaces after commas.

However:

* If every line of code is separated from the next by a blank line, there are too many blank lines.
* If there is neither rhyme nor reason readily discernible for where blank lines are placed, then they are a distraction and there are usually too many of them.
* If a function is so big that it needs many blank lines, it is too big.
* If a block of code needs more than one blank line before or after it, there is something seriously astray.
* If you have more than two blank lines between functions, you probably have too many blank lines.

**Make good use of functions** including parameter passing.

Proper style requires the use of consistent indentation and white space.

You should also ask yourself the following questions:

* Is your solution well designed?
* Did you complete one part of the task before starting another?
* Did you give the user sufficient information?

A rubric containing the marking criteria is located at the end of this document.

Dr. Michael Opoku Agyeman

**Hand in Date: 23/01/2023**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Assessment Item** |  | **Grading Rubric** | | |
| **A** | **B** | **C** | **F** |
| Idea Generation (Design) and Code Functionality (Implementation)  Learning Outcomes: a, b, c, d, e, f | Excellently laid out and prepared portfolio. Demonstrates the ability to communicate clearly and smoothly. Follows the guidelines provided in the assessment brief notes to an outstanding degree. Rational for entry can be easily identified, be understood and includes all required information. Documents and implements more than 80% of full assignment correctly. Implementation and results can easily be repeated by following video tutorial and documentation.  Design specification is logical and coherent. Appropriate methodologies are used, and all elements of these methodologies are skillfully developed.  Exceptional level of development with fully functional and completed outcome. Shows considerable attention to details.  Clear evidence of testing and refinements made. A coherent testing strategy was used.  States a conclusion with justification, based on a reasonable interpretation of the data. Evaluates weaknesses and limitations. Suggests realistic improvements  Show an excellent understanding of the task. Shows an excellent understanding of the way that the technical aspects are combined to produce the system. Exceptionally thorough review and analysis of the underlying theory.  Original &/or distinguished idea, has shown exceptional level of individual competence | Well laid out and prepared portfolio. Demonstrates the ability to communicate clearly and smoothly. Follows the guidelines provided in the assessment brief. Rational for entry can be easily identified, be understood and includes nearly all required information. Documents and implements more than 70% of full assignment correctly. Implementation can be repeated by following video tutorial and documentation.  Design specification is mostly logical and coherent. Appropriate methodologies are used, and some elements of these methodologies are well developed.  High level of development with a mostly completed and functional outcome. Shows a good level of attention to details.  Evidence of testing and refinement made. A coherent testing strategy was used.  States a conclusion based on a reasonable interpretation of the data. Identifies some weaknesses and limitations. Suggests some improvements  Shows proficient understanding of the task.  Shows proficient understanding of how technical aspects are combined to produce the system. Thorough review and analysis of the underlying theory.  Clear and well-developed Statements. | Reasonably well laid out and prepared portfolio. Demonstrates the ability to communicate basically. Rational for entry can be somewhat identified, be understood and includes nearly all required information. Documents and implements more than 60% of full assignment correctly. Implementation can be repeated with some effort by following video tutorial and documentation.  Design specification is reasonable. Some appropriate methodologies are used, and some elements of these methodologies are developed.  Satisfactory level of development with a partially completed and partially functional outcome. Shows a reasonable attention to details.  Satisfactory evidence of testing but not much evidence that refinements were made. A basic testing strategy was used.  States a conclusion with justification, based on a reasonable interpretation of the data. Evaluates weaknesses and limitations. Suggests only superficial improvements.  Shows a good understanding of the task. Makes a sound observation f the problem and is able to analyze some but not all of the more technical aspects of the problem. Good review and analysis of the underlying theory.  Satisfactory and clear, satisfactory level of  individual intellectual capability in dealing with challenges. | Basic or poorly prepared portfolio. Little ability to communicate. Can explain a few of the main ideas, be partially understood and includes a little of the required information. Documents and implements less than 50% of full assignment correctly.  Basic or poor design methodology with little or no effort to use any design methodology.  Basic or poor level of development with an incomplete and partially or non-functional outcome. Shows little or no attention to details.  Little or no evidence of testing and no evidence that refinement was made. No reference to  indicate that a testing strategy was used  States a conclusion which is based on poor interpretation of the data.  Poorly written conclusions Identifies weaknesses which are irrelevant. Suggests irrelevant/no improvements.  Shows basic or no understanding of the task. Doesn’t seem to have a grasp of the problem and is unable to analyze the more technical aspects of the problem. Poor review and analysis of the underlying theory.  Has not presented convincing statements, needs complete rethinking of the statements. |

**Assessment Submission**

To submit your work, please go to the ‘Submit your work’ area of the ***Module*** NILE site. It is important that you submit your work to the correct module NILE site, and that your work is submitted on time.

**Academic Practice**

This is an **individual** assignment and you are expected to work independently. The University of Northampton policy will apply in all cases of copying, plagiarism or any other methods by which students have obtained (or attempted to obtain) an unfair advantage.

Support and guidance on assessments and academic integrity can be found from the following resources

**SkillsHub**: <http://skillshub.northampton.ac.uk>

**CfAP**: <http://tinyurl.com/UoNCfAP>

|  |
| --- |
| **Module Learning Outcome** |
| **Subject-Specific Knowledge, Understanding & Application** |
| 1. Demonstrate reasonably deep issues underlying the design of modern computer architecture |
| 1. Analyse a fairly complex problem and justify the possible solutions |
| 1. Appraise the consequences of certain architectural decisions on the actual performance or power consumption of the processor |
| 1. Evaluate and explain specific tradeoffs between complexity/ cost/performance/power-consumption in modern computer systems |
| **Changemaker & Employability Skills** |
| 1. Be an independent and self-critical learner, guiding the learning of others and managing own requirements for continuing professional development 2. Reflect on own and others' functioning in order to improve practice |