CSSE3010 Project v1.2

Embedded Display Controller for Cellular Automation Graphics

(External Only)

(30% – 60 Marks)

Demo/git Due: Your CSSE 3010 session, Week 13, 2021

Last Updated: May 4, 2021

ECP Hurdle: Must be submitted, to pass.

1 Learning Aims

- Apply your knowledge and skills developed in Stages 1–4 into a new combined design.
- Create a complex embedded system using an RTOS.
- Experience in creating an embedded system with a Top Down Design approach.
- Using peripherals in a modular framework, to implement a complex embedded system.

2 Resources

- Nucleo-F429ZI platform
- Nucleo-F429ZI Red, Green, Blue LEDs
- Nucleo-F429ZI Pushbutton
- LTA1000G LEDBar
- Joystick
- PMOD Keypad
- SSD1306 OLED

In Memoriam, John H. Conway

Mathematician & Game of Life Inventor

December 1937 - April 2020

Contents

3 Introduction

This project will build on your knowledge from stages 1-4. You will be working individually for all tasks. The project is designed to have you demonstrate your knowledge of practical content and ability to integrate new functionality into your project design. This project asks you to develop a Cellular Automation Graphics (CAG) program that simulates and displays simple cellular evolution operations based on the Conway Game of Life. The Conway Game of Life is commonly used to simulate how living organisms can be born, evolve and decay. This project will involve simulating the basic life forms (still, oscillators, spaceships).

The Cellular Automation Graphics (CAG) Controller consists of the following sections:

- CAGSimulator: Control the cellular automation.
- CAGDisplay: Display the output of the CAG.
- CAGKeypad Grid Input: Control the CAGDisplay grid via the keypad.
- CAGKeypad Mnemonic Input: Control the CAGDisplay grid using the commands entered via the keypad.
- CAGJoystick Input: Control the CAG using the the joystick.

Figure 1 shows an overview of the CAG Controller.

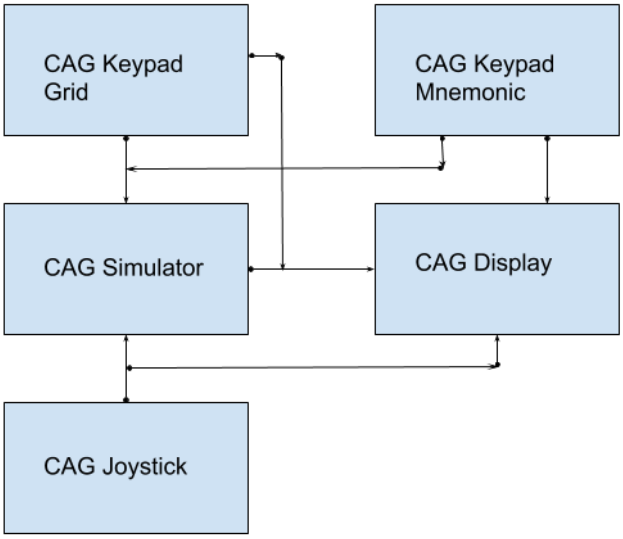

Figure 1: Overview of CAG Overview

4 Design Tasks – 50 marks

4.1 Design Task 1: CAGSimulator – 10 marks

Implement the cellular automation simulator program for the Conway Game of Life. The following cellular evolution rules must be implemented:

1. A live cell with less than two live neighbours will die, due to under-population.
2. A live cell with two or three live neighbours will survive to the next generation.
3. A live cell with more than three live neighbours will die, due to over-population.
4. A dead cell with exactly three live neighbours will become alive. If a dead cell is born, it assumes the highest state value, of the nearest live neighbours.
5. A live cell has to assume the state value of its live neighbours.

Boundary Case

When a cell reaches a boundary or grid edge, then it cannot move any further. The value of cell, will depend on the surrounding cells.

Implementation

The cellular automation library should be implemented as a Task peripheral called `CAG_Simulator` and should conform to the Mylib Task guidelines. Use a FreeRTOS task to implement the Conway Game of Life evolution algorithm. The cellular automation library must use a FreeRTOS queue to receive commands from the keypad interface. A message of the following structure should be used:

```
1 typdef struct caMessage{ int type;   //Type - Cell, or Lifeform
2                     int cell_x;   //Cell x position
3                               //or Lifeform x position
4                     int cell_y;   //Cell y position
5                               //or Lifeform y position
6                     } caMessage_t;
```

The type is an 8-bit value that consist an upper 4-bit nibble (bits 7 to 4) and a lower 4-bit nibble (bits 3 to 0) - see Table 1. Note: x and y positions for cells and lifeforms are the cell x and y coordinates (in the entire grid) and not the sub-grid coordinates. For the lifeforms, you must convert the sub-grid coordinates into cell x and y positions. Figure 2 shows the origin, sub-grid and cell organisation.

FreeRTOS Event Group bits should be used to perform special functions such as:

1. Clear Grid.
2. Start Simulation.
3. Stop Simulation.
4. Set Simulation Update time to 0.5s.

Table 1: Type Values

Type	**Type Bits [7 to 4]**	**Type bits [3 to 0]**
Cell	1	0 - Dead 1 - Alive
Still life	2	0 - block 1 - beehive 2 - loaf
Oscillator	3	0 - blinker 1 - toad 2 - beacon
Space Ship	4	0 - glider

5. Set Simulation Update time to 1s.
6. Set Simulation Update time to 2s.
7. Set Simulation Update time to 5s.
8. Set Simulation Update time to 10s.

The LEDBar[9] segment must be toggled when a Simulator update or Simulation cycle has occurred (e.g. new grid has been calculated).

All CAG_Simulator Mylib Task files must be placed in `mylib` folder.

4.2 Design Task 2: CAGDisplay – 10 marks

The Display output must be implemented using the SSD1306 OLED Display. The SSD1306 OLED can display 128 by 32 pixels. Each cell should map to a 2 by2 pixel square.
8/2
Please refer to the `examples/peripheral/oled` example.

Grid Display

The grid display consists of sub-grids and smaller cells. A sub-grid consists of a grid of `3 x 3` cells. The sub-grid is accessed by the keypad in grid mode. Figure 2 shows the origin, sub-grid and cell organisation.

The grid display features are shown in Table 2.

Implementation

You must use a task to control the CAGDisplay output. Create a `CAG_Display` Mylib Task files in your `mylib` folder. You must control your SSD1306 OLED display output using FreeRTOS elements (e.g. queues, semaphores, etc). FreeRTOS Event Group bits should be used to perform special functions such as:

1. Set Alive Cell Colour to Black

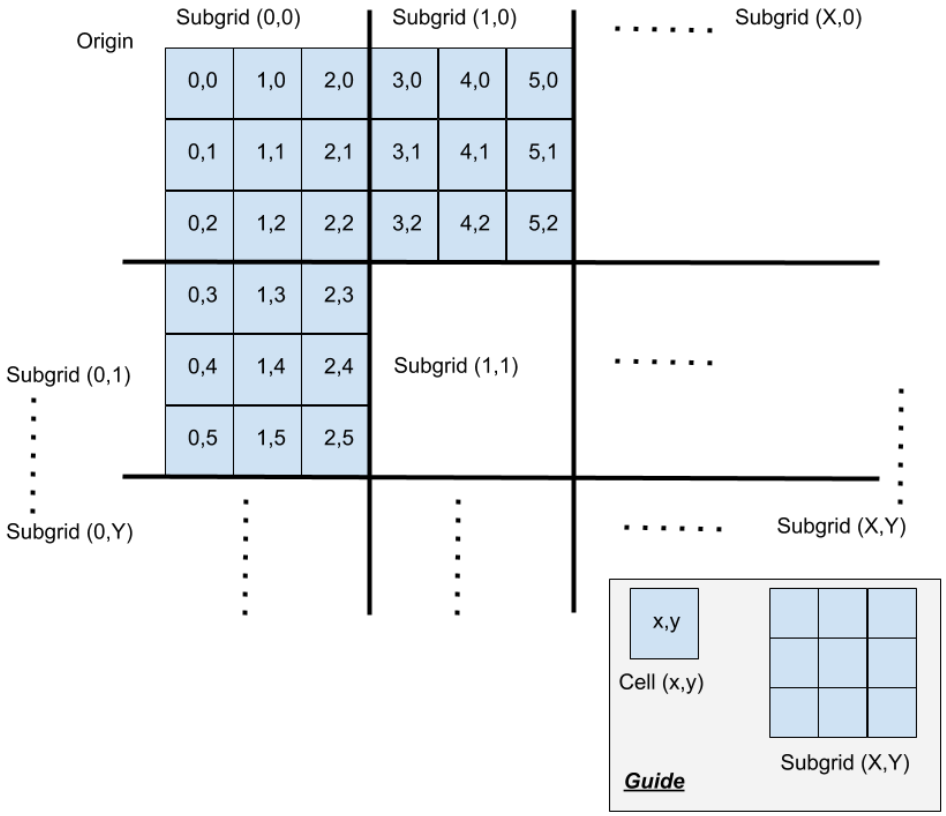

Figure 2: Overview of CAGGrid Display showing sub-grids and Cells)

Table 2: Grid Display Parameters

Parameter	**Description**	**Setting**
Cell	Grid Cell	A cell should be a size of 2 to 4 ASCII characters wide and 2 to 4 lines in height.
sub-grid	sub-grid	A $3x3$ grid of cells
Grid Width	Total Width of Grid	30 cells (5 sub-grids) wide.
Grid Height	Total Height of Grid	15 cells (5 sub-grids) high.

2. Set Alive Cell Colour to Red
3. Set Alive Cell Colour to Green
4. Set Alive Cell Colour to Yellow
5. Set Alive Cell Colour to Blue
6. Set Alive Cell Colour to Magenta
7. Set Alive Cell Colour to Cyan
8. Set Alive Cell Colour to White

All CAG_Display Mylib Task files must be placed in `mylib` folder.

4.3 Design Task 3: CAGKeypad - Grid Mode – 10 marks

The PMOD keypad must be used to control the CAGdisplay. The keypad must operate in two different modes: Grid and Mnemonic. The keypad mode must be toggled by the onboard user pushbutton (no other button is permitted). Default mode is grid. The onboard red LED must be on when in grid mode otherwise the onboard blue LED must be on, when in mnemonic mode.

In grid mode, the keypad allows individual cells to be set, within the selected sub-grid . Table 3 shows the function of each key. A sub-grid consists of `3 x 3` grid of cells. Cell keys will set the values within a sub-grid . The cell keys are used to set the state of a cell. The cell key should toggle cell to be alive or dead (default state is dead). The selection of the sub-grid is set the sub-grids ' X and Y key presses. Each sub-grid key press should increment counters for sub-grids ' X or Y (loops back). The LEDBar must show the current sub-grids ' X (LED[5:3]) and Y (LED[2:0]) values.

E.g. Pressing the sub-grid X key 3 times and the sub-grid Y key 2 times will refer to the sub-grid at position (3,2). And pressing the X key an additional 2 times, will the sub-grid to position (0, 2).

Table 3: Grid Key Assignment

1 Cell e.g. [0,0]	**2** Cell e.g. [1,0]	**3** Cell e.g. [2,0]	**A** Start
4 Cell e.g. [0,1]	**5** Cell e.g. [1,1]	**6** Cell e.g. [2,1]	**B** Stop
7 Cell e.g. [0,2]	**8** Cell e.g. [1,2]	**9** Cell e.g. [2,2]	**C** Clear
0 sub-grid X	**F** sub-grid Y	**E** Not used	**D** Not used

MyLib Task Implementation

You must write an Mylib Task driver for the keypad, which must be located in the `mylib` folder. Each key should be mapped to an Event Group Bit. **When any key is pressed, the onboard green LED must be toggle**.

You must use a task to control the keypad grid mode. Create a `CAG_keypad_grid` Mylib Task files in your `mylib` folder. Use FreeRTOS elements (e.g. queues, semaphores, etc). The `CAG_keypad_grid` must use keypad Mylib Task driver. A semaphore should be used to control the mode of the keypad. The `CAG_keypad_grid` must control the `CAG_Simulator` mylib Task driver.

All CAG_keypad_grid Mylib Task files must be placed in `mylib` folder.

4.4 Design Task 4: CAGKeypad - Mnemonic Mode – 10 marks

The PMOD Keypad must be used to control the CAGdisplay. The keypad must operate in two different modes: Mnemonic and Grid. The keypad mode must be toggled by the onboard user pushbutton (no other button is permitted). In mnemonic mode, the keypad allows commands or mnemonics to control the CAGDisplay. The mnemonics commands are shown in Table 4.

Table 4: Mnemonic Commands

Mnemonic Command	**Parameters**	**Description**
`STL`	<type><x><y>	Draw a still life at sub-grid (x, y). Types: block (0), beehive(1), loaf(2).
`OSC`	<type><x><y>	Draw an oscillator at sub-grid (x, y). Types: blinker (0), toad(1) and beacon(2).
`GLD`	<x><y>	Draw a glider at sub-grid (x, y).
*		Start or restart CAG_Simulator.
#		Stop CAG_Simulator.
`0`		Clear CAG_Display.
`DLT`	<type>	Delete the specified specified driver (0) CAG_Simulator (1) CAG_Joystick
`CRE`	<type>	Create the specified specified driver (0) CAG_Simulator (1) CAG_Joystick

The keypad should be configure to a ITU E 1.161 telephone alphanumeric keypad input. Only the alphanumeric input is required (0-9 and A to Z) with * and # keys. Use 'F' as * and 'D' as #. Multiple key presses will be required to select a letter.

MyLib Task Implementation

You use the Mylib Task driver for the keypad, which must be located in the `mylib` folder. See section 4.3.

You must use a task to control the keypad mnemonic mode. Create a `CAG_keypad_mnemonic` Mylib Task files in your `mylib` folder. Use FreeRTOS elements (e.g. queues, semaphores, etc). The `CAG_keypad_mnemonic` must use keypad Mylib Task driver. A semaphore should be used to control the mode of the keypad. The `CAG_keypad_mnemonic` must control the `CAG_Simulator` mylib Task driver.

All CAG_keypad_mnemonic Mylib Task files must be placed in `mylib` folder.

4.5 Design Task 5: CAGJoystick Control – 10 marks

The Joystick X and Y signals are used to control the colour of alive cells shown by the CAGDisplay and refresh rates of the CAGSimulator. The Joystick Z signal is used to clear the CAGDis-

play, when pressed.

The Joystick X signal must control the colour of the alive cells with the minimum extreme set to black and the mid-point (upright joystick) to the maximum extreme is set to white.

The Joystick Y signal must control the simulation update time of the CAGsimulator with the minimum extreme set to 0.5s, mid-point (upright joystick) set to 2s and the maximum extreme is set to 10s. Other values must be selected between the minimum and maximum extremes. The simulation update rate times must not be varied, other than those specified.

MyLib Task Implementation

You must write an Mylib Task driver for the joystick, which must be located in the `mylib` folder. Joystick X/Y signals should be accessed via queue. The Joystick Z signal should be accessed via Semaphore.

You must create a CAG_joystick mylib Task driver that uses the Joystick Mylib Task Driver. The CAG_joystick Mylib Task driver must control the CAGDisplay and CAGSimulator.

All CAG_joystick Mylib Task files must be placed in the `mylib` folder.

5 Coding Practices – 10 marks

5.1 Code Quality – 2 marks

It is imperative that code you write be able to be easily understood and modified/expanded by your peers and colleagues. As such, your code will be reviewed by a tutor. Aspects of high code quality include (but are not limited to):

- descriptive, readable naming of variables, functions, parameters, constants, macros, types
- constants and defines used instead of "magic values"
- modularity and functional decomposition instead of repeated code
- consistent white space (horizontal and vertical)
- conformation with the CSSE3010 Style Guide (on Blackboard)
- debugging and testing code neatly excluded from release compile for submission
- design justifications where appropriate
- documentation for functions
- comments for global variables, constants, macros, etc where appropriate

Note that you are encouraged to have debug/test code and extra functionality where appropriate. #define and build configurations should be used to enable or disable the debug/test code. Commented out code is not permitted.

5.2 Code Structure – 8 marks

This project has the following requirements:

1. Implement using FreeRTOS.
2. Implement using `Sourcelib`.
3. Must be implemented using your mylib library.
4. Git Version Control – You must use git as part of your development process.

myLib Requirements

You are required to use your mylib library files in this project. Your mylib and project code must conform to the following structure, shown in Table 5, in order for your project to be marked.

Table 5: Folder Structure

Git Folder	Sub-folders	Notes
`mylib`	No sub folders	All _lib, _tsk and _reg files that are used and can be compiled without errors.
`stages`	`pf`	Must contain your `main.c` and other supporting files (e.g. `FreeRTOSConfig.h`, `it_handler.c`)

Only files in the specified folders will be marked and used to build your project `.bin` file. No other files, other than `sourcelib`, will be used..

main.c

Functions called in main.c must only be used to initialise the system, not to control the system. Design task implementations should be done with tasks, using task mylib files. Your `main.c` file must only be used to create only mylib tasks and to start the FreeRTOS Scheduler. No hardware initialisation or other functions like interrupt routines or tasks should be present in `main.c`.

FreeRTOS

You must demonstrate in your code, that you have used the features provided by FreeRTOS. Tasks MUST be used to implement various features of the system.

Interrupt Service Routine

Interrupt Service Routines (ISR) must not be used to directly implement the design tasks. ISRs should control a task with queues, semaphores, events, etc.

All interrupt service routine handlers MUST be in your `it_handler.c` file

6 Marking

Marked in your session, in week 13.

6.1 Final Demo Video Upload

By the project due date, you must upload a short (max 60s) video or screen capture showing your display output working with a simple oscillator or glider spaceship being simulated. You must first show your student card, before showing your screen output. The video must be upload to BlackBoard (BlackBoard $->$ Assessment$->$Project$->$Project Video upload).

If your project has no functionality or there is no CAGDisplay does not work, then a video should not be uploaded.

If you project does have a functioning (semi or complete) CAGDisplay output and you do not upload a video, then your project will NOT be assessed.

6.2 Milestone and Final Project Git Version Control

NOTE: Version Control – You MUST use version control as part of your development process. This means that because the project is a bigger task than the stages, you should have at least 3 commits (added from the Milestone due date) by the Project due date, otherwise your project will not be assessed.

Each task is allocated a number of marks. Full marks will be awarded to seamless, well integrated tasks with no errors. Part marks may be awarded for partially working solutions. You must be able to demonstrate functionality to achieve marks.

6.3 Deductions

The following situations will result in the allocated deduction. Deductions can only be deducted a single time. If you have questions, talk to course staff.

Deduction	Reason
8	Not building on the work done in stages e.g. not using mylib structure specified in stages.
4	main.c contains more than initialisation code.
4	Excessively unreadable code (no conformity to mylib or general style guide).
4	Significant disregard or no conformity to the mylib or general style guide).
4	Multiple reprogramming of the Nucleo at any point throughout the demo, to observe design tasks.
2	Excessive resetting of the Nucleo is required to observe various design tasks
2	Commented out code