The University of Queensland
School of Information Technology and Electrical Engineering
Semester 1, 2021

CSSE2010 / CSSE7201 Lab Assignment 2

Due: **4:00pm Friday May 28, 2021**
Weighting: **20% (100 marks)**

Objective

As part of the assessment for this course, you are required to undertake a project which will test you against some of the more practical learning objectives of the course. The project will enable you to demonstrate your understanding of

- C programming
- C programming for the AVR
- The Microchip Studio environment.

You are required to modify a program in order to implement additional features. The program is a basic template of the board game Reversi (a description is given on page 3).
For FD students: the AVR ATmega324A microcontroller runs the program and receives input from a number of sources and outputs a display to an LED display board, with additional information being output to a serial terminal and – to be implemented as part of this project – a seven segment display and other LEDs.
For EX students: the AVR ATmega328P microcontroller runs the program and receives input from a number of sources and outputs a display to a serial terminal and – to be implemented as part of this project – a seven segment display and other LEDs.

The version of Reversi provided to you has very basic functionality – it will present a start screen upon launch, respond to button presses or a terminal input 's' to start the game, then display the starting board for the game Reversi with a cursor that flashes on and off. You can add features such as moving the cursor, placing pieces, scoring, pausing, detecting legal moves, sound effects, etc. The different features have different levels of difficulty and will be worth different numbers of marks.

Don't Panic!

You have been provided with approximately 2000 lines of code to start with – many of which are comments. Whilst this code may seem confusing, you don't need to understand all of it. The code provided does a lot of the hard work for you, e.g., interacting with the serial port and the LED display. To start with, you should read the header (.h) files provided along with game.c and project.c. You may need to look at the AVR C Library documentation to understand some of the functions used.

Academic Merit, Plagiarism, Collusion and Other Misconduct

You should read and understand the statement on academic merit, plagiarism, collusion and other misconduct contained within the course profile and the document referenced in that course profile. **You must not show your code to or share your code with any other student under <u>any</u> circumstances. You must not post your code to public discussion forums or save your code in publicly accessible repositories. You must not look at or copy code from any other student. All submitted files will be subject to electronic plagiarism detection and misconduct proceedings will be instituted against students where plagiarism or collusion is suspected.** The electronic plagiarism detection can detect similarities in code structure even if comments, variable names, formatting etc. are modified. If you copy code, you will be caught.

Grading Note

As described in the course profile, if you do not score at least 15% on this project (before any late penalty) then your course grade will be capped at a 3 (i.e. you will fail the course). If you do not obtain at least 50% on this project (before any late penalty), then your course grade will be capped at a 5. Your project mark (after any late penalty) will count 20% towards your final course grade. Resubmissions prior to grade finalization are possible to meet the 15% requirement in order to pass the course, but a late penalty will be applied to the mark for final grade calculation purposes.

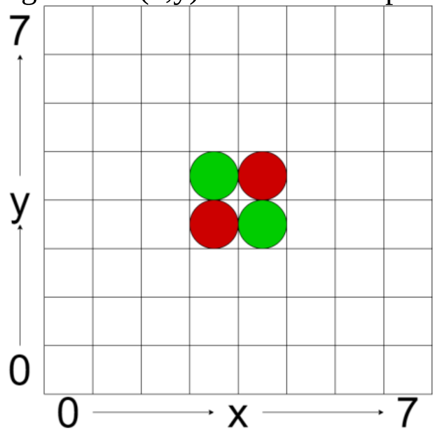
Program Description

The program you will be provided with has several C files which contain groups of related functions. The files provided are described below. The corresponding .h files (except for project.c) list the functions that are intended to be accessible from other files. You may modify any of the provided files. You must submit ALL files used to build your project, even if you have not modified some provided files. Many files make assumptions about which AVR ports are used to connect to various IO devices. You are encouraged not to change these.

- **project.c** – this is the main file that contains the event loop and examples of how time-based events are implemented. You should read and understand this file.
- **game.h/game.c** – this file contains the implementation of the board used to store the state of the game and the position of the cursor. You should read this file and understand what representation is used for the board state and the cursor position. You will need to modify this file to add required functionality.
- **display.h/display.c** – this file contains the implementation for displaying the current state of the board. This file contains useful functions for displaying the board to the LED matrix (flexible students) or the terminal display (external students). This file contains the same functions for FD and EX students but with significantly different implementations.
- **buttons.h/buttons.c** – this contains the code which deals with the IO board push buttons. It sets up pin change interrupts on those pins and records rising edges (buttons being pushed).
- **ledmatrix.h/ledmatrix.c** (FD students only) – this contains functions which give easier access to the services provided by the LED matrix. It makes use of the SPI routines implemented in spi.c
- **pixel_colour.h** (FD students only) – this file contains definitions of some useful colours
- **serialio.h/serialio.c** – this file is responsible for handling serial input and output using interrupts. It also maps the C standard IO routines (e.g. printf() and fgetc()) to use the serial interface so you are able to use printf() etc for debugging purposes if you wish. You should not need to look in this file, but you may be interested in how it works and the buffer sizes used for input and output (and what happens when the buffers fill up).
- **spi.h/spi.c** (FD Students only) – this file encapsulates all SPI communication. Note that by default, all SPI communication uses busy waiting (i.e. polling) – the "send" routine returns only when the data is sent. If you need the CPU cycles for other activities, you may wish to consider converting this to interrupt based IO, similar to the way that serial IO is handled.
- **terminalio.h/terminalio.c** – this encapsulates the sending of various escape sequences which enable some control over terminal appearance and text placement – you can call these functions (declared in terminalio.h) instead of remembering various escape sequences. Additional information about terminal IO will be provided on the course Blackboard site.
- **timer0.h/timer0.c** – sets up a timer that is used to generate an interrupt every millisecond and update a global time value.

Reversi Description

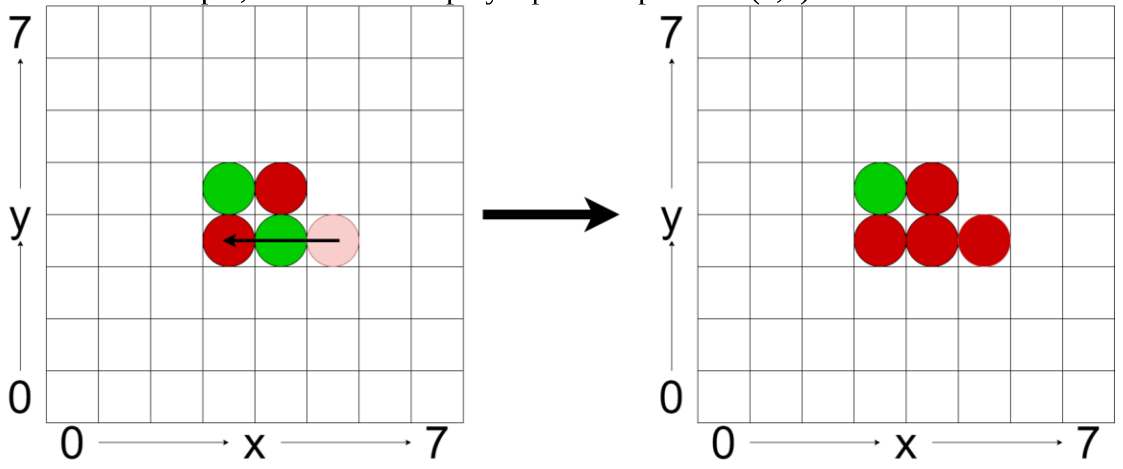
This project involves creating a replica of the board game 'Reversi'. Reversi is a turn-based game played between two players on an 8x8 board which starts with 4 pieces positioned as below. In the supplied code, each square is given an (x,y) coordinate as pictured.



On each player's turn, they play one piece of their colour and then flip some of the opponent's pieces (described below) to be their colour. The game ends when all squares have been filled (or neither player can make a move) and the winner is the player who has more of their coloured pieces on the board, if both players have equally many pieces the game is declared a draw.

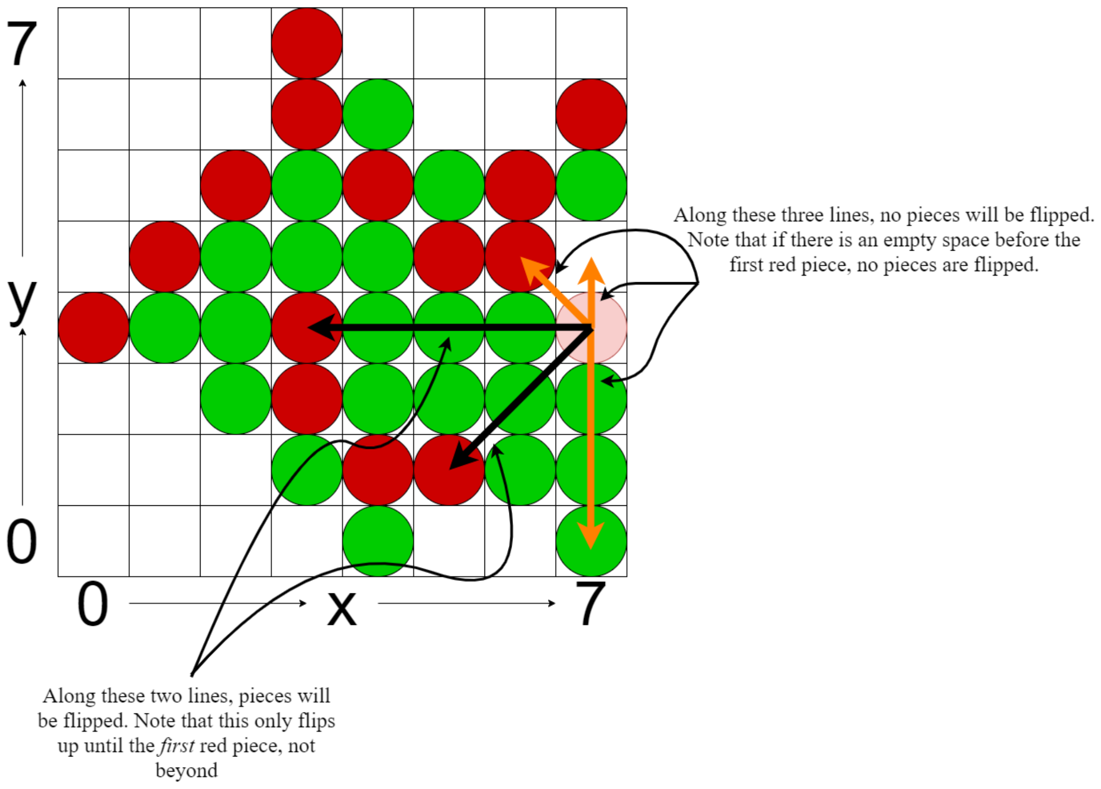
Flipping Opponent's Pieces

Red has the first turn, and on red's turn they place a red piece. After a piece is played, in all 8 directions (horizontal, vertical and diagonal) from where the piece is played, if there are one or more green pieces followed by a red piece, then those green pieces are flipped over and become red pieces. For example, if red chose to play a piece at position (5,3):

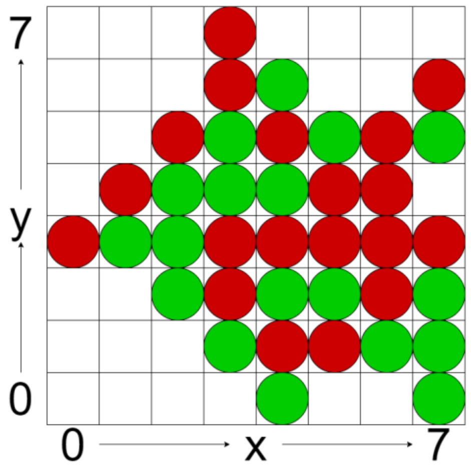


Then along the straight line horizontally to the left, there is a green piece followed by a red piece, so the green piece is flipped and becomes a red piece. The rules for placing a green piece are the same but with the opposite colours. For a move to be legal, it must flip at least one of the opponent's pieces.

An example placement of a red piece at position (7,3) later in a game is shown below:


7
y
0
0
x
7
Along these three lines, no pieces will be flipped. Note that if there is an empty space before the first red piece, no pieces are flipped.
Along these two lines, pieces will be flipped. Note that this only flips up until the *first* red piece, not beyond

After this piece is played, the board has the following state:



On either player's turn, if there are no legal moves available, their turn is passed, and the active player returns to the other player. If neither player has any legal moves available, then the game is over and the player with more pieces is declared the winner.

To check your game is working as expected, you may use an online Reversi game such as this (Cool Math Games, 2021).

Initial Operation

The provided program has very limited functionality. It will display a start screen which detects the rising edge on the buttons connected to pins B0, B1 and B2, and also the input terminal character 's'. Pressing of any of these will start a game with the start configuration and a flashing cursor.

Once started, the program detects a rising edge on the button connected to pin B2, but no action is taken on this input.

Wiring Advice

When completing this assignment, you will need to make additional connections to the ATmega324A/ ATMega328P. To do this, you will need to choose which pins to make these connections to. For FD students, there are multiple ways to do this, so the exact wiring configuration will be left up to you, and you should communicate this using your submitted feature summary form.

Due to the reduced number of external pins available with the Arduino Uno for EX students, all pins have to be used to add all features and working out a valid configuration could be quite tricky, as a result a configuration is provided below which you are encouraged to use to ensure everything fits. Note that pins D1 and D0 are the TX/RX pins, which are being used to communicate to the serial terminal and no connections should be made to them.

For EX Students Only

There was a mistake in the initially given wiring advice that placed the piezo buzzer on B1 (OC1A) instead of B2 (OC1B). The initially given wiring advice will still work if you attempt everything except "Sound Effects" but will not be able to be used if you do attempt "Sound Effects". As such the wiring advice has been updated as below. This does change the SSD connections to be split into a 6-block and a 2-block.

After conducting the external practicals, we found that there was quite a bit of inconsistency between devices and circuit behaviour when using the Seven Segment Display and NOT gate transistor circuit. Because of this, we encourage you to instead use two CC pin connections to the Arduino, one for each CC pin on the Seven Segment Display. To wire this up, connect each CC pin from the Seven Segment Display to the relevant Arduino pin via a 330 Ohm resistor (if you do not have any spare 330 Ohm resistors, take them from the LEDs). All other connections are as normal. When programming the Seven Segment Display, note that you will need to control both CC pins in software, and ensure they are always opposite one another when writing individual digits on the SSD to replicate the NOT gate behaviour.

Because of this change, we have also released an updated solution for lab13-2 which uses an SSD configuration more closely resembling that required for this project. It has the SSD connections split into A-F and G-DP, as well as two CC connections instead of 1. <This code will still require adaptation for the project but should be a good starting point.>

Old (INCORRECT) Advice

Port	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1	Pin 0
B			SSD A-D				Piezo	SSD CC1
C			ADC connections		SSD CC2 (if needed)	Button B2	Button B1	Button B0
D	SSD E-DP				LEDs		Reserved for RX/TX Baud rate: 19200	

New (CORRECT) Advice

Port	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1	Pin 0
B			SSD DP	SSD G	SSD CC1	Piezo	LEDs	
C			ADC connections		SSD CC2	Button B2	Button B1	Button B0
D	SSD A-F						Reserved for RX/TX Baud rate: 19200	

Program Features

Marks will be awarded for features as described below. Part marks will be awarded if part of the specified functionality is met. Marks are awarded only on demonstrated functionality in the final submission – no marks are awarded for attempting to implement the functionality, no matter how much effort has gone into it, unless the feature can be demonstrated. You may implement higher-level features without implementing all lower level features if you like (subject to prerequisite requirements). The number of marks is **not** an indication of difficulty. It is much easier to earn the first 50% of marks than the second 50%.

You may modify any of the code provided and use any of the code from learning lab sessions and/or posted on the course Blackboard site. For some of the easier features, the description below tells you which code to modify or there may be comments in the code which help you.

Minimum Performance (Level 0 – Pass/Fail)

Your program must have at least the features present in the code supplied to you, i.e., it must build and run, show the start screen, and display the initial board when a button or 's' is pressed. No marks can be earned for other features unless this requirement is met, i.e., your project mark will be zero.

Start Screen (Level 1 – 4 marks)

Modify the program so that when it starts (i.e. the AVR microcontroller is reset) it outputs your name and student number to the serial terminal. Do this by modifying the function `start_screen()` in file *project.c*.

Move Cursor with Buttons (Level 1 – 12 marks)

The provided program does not allow for the cursor to be moved. Modify the program so that button B2 (connected to pin B2 for FD students, connected to pin C2 for EX students) moves the cursor to the left and button B1 moves the cursor upwards. Note that there is no button to move downwards or to the right. If the cursor moves off the display, it should wrap around to the opposite side. For example, if the cursor is at square (5,7) and attempts to move upwards, it should wrap to the bottom to square (5,0). When the cursor moves, it should be visible immediately and should restart its flashing cycle.

In the `play_game()` function in the file *project.c*, when button 2 is pressed, the function `move_display_cursor(dx,dy)` in the file *game.c* is called. This function is currently empty, start by filling in the `move_display_cursor` function, there are some hints to get you started. Then update `play_game` to detect button B1 as well.

Move Cursor with Terminal Input (Level 1 – 5 marks)

The provided program does not register any terminal inputs once the game has started. Modify the program such that pressing 'W' moves the cursor upwards, 'A' moves the cursor to the left, 'S' moves the cursor downwards and 'D' moves the cursor to the right. Both the lower case and upper case of each letter should move the cursor.

On the start screen, the game can be started by pressing 's' or 'S', looking at the function `start_screen()` should give you an idea of how to read serial input from the terminal.

Note that if you press the directional arrows, they might also move your cursor in an unexpected direction, this is because they use escape characters that contain 'A' and 'D'. We will not be considering escape characters when testing and will only use the letters of the alphabet, digits and spacebar as potential inputs.

Piece Placement #1 (Level 1 – 8 marks)

(This assumes that you have implemented "Move Cursor with Buttons" above.) Modify the program so that a piece can be placed at the current location of the cursor when button B0 or spacebar is pressed. For this level, the piece played does not have to be a legal move and does not have to flip any of the opponent's pieces (this will come later, see Piece Placement 2, 3 & 4) it should simply place a piece at the current location. When a piece is played, the current player should switch (it starts as player 1/red).

Scoring #1 **(Level 1 – 8 marks)**

(This assumes that you have implemented "Piece Placement #1" above.) Display the current score of both players to the terminal in a fixed position. Add appropriate text (e.g "Red Score:" and "Green Score:") to the terminal display so it is clear where the score is displayed and which score refers to which player. Update the score display only when it changes. The displayed scores must be right-aligned – i.e. the right-most digit in each score must be in a fixed position. (The score need not be on the right-hand edge of the terminal display – it just must be right-aligned within a given field position – i.e. the least significant digit must always be in the same location.) A score of 2-2 must be displayed when the game starts. The score must remain displayed on game-over (until a new game commences when the score should be reset).

Scoring #2 **(Level 1 – 10 marks)**

(Assumes that "Scoring #1" is implemented.) Display the score of the current player on the seven segment display. For scores from 0 to 9 inclusive the left seven segment digit should be blank. No display flickering should be apparent. Both digits (when displayed) should be of equal brightness. Include any connections required on your submitted feature summary form.

LED Turn Display **(Level 1 – 6 marks)**

(Assumes that "Piece Placement #1" is implemented.) Display which player's turn it is using LEDs L2 (green) and L1 (red). On the red player's turn, the red LED should be lit, on the green player's turn, the green LED should be lit.

For external students, if you do not have different coloured LEDs, that is okay. Just light up two different LEDs for each player's turn and include on your feature summary form which LED is for player 1 and which LED is for player 2.

Piece Placement #2 **(Level 2 – 10 marks)**

(Assumes that "Piece Placement #1" is implemented.) For any legal move made, flip all the required opposition pieces to be this colour. This should follow the logic for flipping opposition pieces as outlined in the Reversi Description above. For this part, the behaviour must only be correct if the piece being played is being played in a legal position, there is no requirement for what happens if the piece is played in an illegal position, that will come in Piece Placement #3.

Piece Placement #3 **(Level 2 – 10 marks)**

(Assumes that "Piece Placement #2" is implemented.) Placement of a piece at a square is illegal if placing it there would result in no opposition pieces being flipped. When the cursor is on an illegal square, this should be displayed by changing the colour of the cursor (to a colour different to all existing colours on the display). If an attempt to play a piece (spacebar or button B0) is made at an illegal position, the move should be rejected, and the current player should not change.

Game Pause **(Level 2 – 6 marks)**

Modify the program so that if the 'p' or 'P' key on the serial terminal is pressed then the game will pause. When the button is pressed again, the game recommences. (All other button/key presses/inputs should be discarded whilst the game is paused – i.e. the cursor cannot be moved, and no pieces can be placed.) The cursor flashing must be unaffected – e.g. if the pause happens 200ms before the cursor is going to flash off, then the cursor should not flash off until 200ms after the game is resumed, not immediately upon resume.

Turn Timing **(Level 2 – 6 marks)**

(Assumes that "Piece Placement #1", "Scoring #2" and "LED Turn Display" are implemented.) Adds the ability to toggle between a "timed game" and a "non-timed game" (default) by pressing the 't' or 'T' key on the serial terminal. In a timed game, if either player takes 30 seconds to make a move, then they forfeit the game and the game ends, with the other player being declared the winner by a score of 64-0. If thirty seconds elapses since the start of either player's turn, the game should end and the score 64-0 displayed.

During a timed game, the time remaining (in seconds) should be displayed on the seven segment display. This will replace the scoring display (described in "Scoring #2") during a timed game but should not alter the display otherwise. i.e when playing a non-timed game, the current player's score should still be displayed.

A game may be changed to a timed game at any point by pressing 't' or 'T', with the 30 second counter starting immediately. Any time the game is switched to a timed game, the timer starts from the full 30 seconds. A timed game may be stopped at any point by pressing 't' or 'T'

Piece Placement #4 (Level 3 – 4 marks)

(Assumes that "Piece Placement #3" is implemented.) If there are no legal moves available for a player, then they are forced to pass their turn. For example, if red plays a piece at (7,6) then there are no legal moves for green to make, then green passes their turn and it is red's turn again. If neither player can make a move, the game is over. Modify your program to add this behaviour.

Joystick (Level 3 – 5 marks)

Add support to use the joystick to move the cursor left, right up and down. This must include support for auto-repeat – if the joystick is held in one position then, after a short delay, the code should act as if the joystick was repeatedly moved in that direction. Your movement must be reasonable – i.e a reasonable player could stop at whichever square they desired. Your cursor must be shown to move through all positions and be able to be stopped at that position if the joystick is released. Be sure to specify which AVR pins the U/D and L/R outputs on the joystick are connected to.

Be aware that different joysticks may have different min/max/resting output voltages and you should allow for this in your implementation – your code will be marked with a different joystick to the one you test with. For external students, this will be marked more leniently as your joystick likely differs more substantially.

Sound Effects (Level 3 – 6 marks)

Add sound effects to the program which are to be output using the piezo buzzer. Different sound effects (tones or combinations or tones) should be implemented for at least four events. (At least two of these must be sequences of tones for full marks.) For example, choose four of:

- Cursor being moved
- Legal move being made
- Illegal move being made
- Game start-up
- Constant background tune

Do not make the tones too annoying! Pressing the 'm' or 'M' key on the serial terminal must be used to toggle sound on and off, sound should be on when the game starts. You must specify which AVR pin the piezo buzzer must be connected to. (The piezo buzzer will be connected from there to ground.) Your feature summary form must indicate which events have different sound effects. Sounds must be tones (not clicks) in the range 20Hz to 5kHz. Sound effects must not interfere with game play, e.g. the speed of play should be the same whether sound effects are on or off. Sound must turn off if the game is paused.

Assessment of Program Improvements

The program improvements will be worth the number of marks shown above. You will be awarded marks for each feature up to the maximum mark for that feature. Part marks will be awarded for a feature if only some part of the feature has been implemented or if there are bugs/problems with your implementation (which may include issues such as incorrect data direction registers). Your additions to the game must not negatively impact the playability or visual appearance of the game. Note also that the features you implement must appropriately work together, for example, if you implement game pausing then sound effects should pause.

Features are shown grouped in their levels of approximate difficulty (level 1, level 2, and level 3).

Submission Details

The due date for the project is **4:00pm Friday 28 May 2021**. The project must be submitted via Blackboard. You must **electronically submit a single .zip** file containing ONLY the following:

- **All** of the C source files (.c and .h) necessary to build the project (including any that were provided to you – even if you haven't changed them);
- Your final .hex file (suitable for downloading to the ATmega324A/ATmega328P AVR microcontroller program memory); and
- A PDF feature summary form (see below).

Do not submit .rar or other archive formats – the single file you submit must be a zip format file. All files must be at the top level within the zip file – do not use folders/directories or other zip/rar files inside the zip file.

If you make more than one submission, each submission must be complete – the single zip file must contain the feature summary form, the hex file and all source files needed to build your work. We will only mark your last submission and we will consider your submission time (for late penalty purposes) to be the time of submission of your last submission.

The feature summary forms are on the last pages of this document, there is a separate feature summary form for FD and EX students. A separate electronically-fillable PDF form will be provided to you also. This form can be used to specify which features you have implemented and how to connect the ATmega324A/ATmega328P to peripherals so that your work can be marked. If you have not specified that you have implemented a particular feature, we will not test for it. Failure to submit the feature summary with your files may mean some of your features are missed during marking (and we will NOT remark your submission). You can electronically complete this form or you can print, complete and scan the form. Whichever method you choose, you must submit a PDF file with your other files.

Incomplete or Invalid Code

If your submission is missing files (i.e. won't compile and/or link due to missing files) then we will substitute the original files as provided to you. No penalty will apply for this, but obviously no changes you made to missing files will be considered in marking.

If your submission does not compile and/or link in Microchip Studio for other reasons, then the marker will make reasonable attempts to get your code to compile and link by fixing a small number of simple syntax errors and/or commenting out code which does not compile. **A penalty of between 10% and 50% of your mark will apply depending on the number of corrections required.** If it is not possible for the marker to get your submission to compile and/or link by these methods then you will receive 0 for the project (and will have to resubmit if you wish to have a chance of passing the course). A minimum 10% penalty will apply, even if only one character needs to be fixed.

Compilation Warnings

If there are compilation warnings when building your code (in Microchip Studio, with default compiler warning options) then a mark deduction will apply – **1 mark penalty per warning up to a maximum of 10 marks.** To check for warnings, rebuild ALL of your source code (choose "Rebuild Solution" from the "Build" menu in Microchip Studio) and check for warnings in the "Error List" tab.

General Deductions

If there are problems in your submitted project that do not fit into any of the above feature categories, then some marks may be deducted for these problems. This could include problems such as general lag, errors introduced to the original program etc.

Late Submissions

Late submission will result in a penalty of 10% plus 10% per calendar day or part thereof, i.e. a submission less than one day late (i.e. submitted by 4:00pm Saturday 29 May, 2021) will be penalised 20%, less than two days late 30% and so on. (The penalty is a percentage of the mark you earn (after any of the other penalties described above), not of the total available marks.) Requests for extensions should be made via the process described in the course profile (before the due date) and be accompanied by documentary evidence of extenuating circumstances (e.g. medical certificate). The application of any late penalty will be based on your latest submission time.

Notification of Results

Students will be notified of their results via Blackboard's "My Grades".

The University of Queensland – School of Information Technology and Electrical Engineering
Semester 1, 2021 – CSSE2010 / CSSE7201 Project – Feature Summary EXTERNAL

Student Number								Family Name	Given Names

An electronic version of this form will be provided. You must complete the form and include it (as a PDF) in your submission. You must specify which IO devices you've used and how they are connected to your ATmega328P.

Port	**Pin 7**	**Pin 6**	**Pin 5**	**Pin 4**	**Pin 3**	**Pin 2**	**Pin 1**	**Pin 0**
B	■	■						
C	■	■				Button B2	Button B1	Button B0
D							Reserved for RX/TX Baud rate: 19200	

Feature	**✓ if attempted**	**Comment** **(Anything you want the marker to consider or know?)**	**Mark**	
Start screen			/4	
Move Cursor with Buttons			/12	
Move Cursor with Terminal Input			/5	
Piece Placement #1			/8	
Scoring #1			/8	
Scoring #2			/10	
LED Turn Display			/6	/53
Piece Placement #2			/10	
Piece Placement #3			/10	
Turn Timing			/6	
Game Pause			/6	/32
Piece Placement #4			/4	
Joystick			/5	
Sound Effects			/6	/15

Total: (out of 100)	
General deductions: (errors in the program that do not fall into any above category, e.g general lag in gameplay)	
Penalties: (code compilation, incorrect submission files, etc. Does not include late penalty)	
Final Mark: (excluding any late penalty which will be calculated separately)	

The University of Queensland – School of Information Technology and Electrical Engineering
Semester 1, 2021 – CSSE2010 / CSSE7201 Project – Feature Summary FLEXIBLE

Student Number								Family Name	Given Names

An electronic version of this form will be provided. You must complete the form and include it (as a PDF) in your submission. You must specify which IO devices you've used and how they are connected to your ATmega324A.

Port	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1	Pin 0
A								
B	SPI connection to LED matrix					Button B2	Button B1	Button B0
C								
D							Serial RX Baud rate: 19200	Serial TX

Feature	✓ if attempted	Comment (Anything you want the marker to consider or know?)	Mark	
Start screen			/4	
Move Cursor with Buttons			/12	
Move Cursor with Terminal Input			/5	
Piece Placement #1			/8	
Scoring #1			/8	
Scoring #2			/10	
LED Turn Display			/6	/53
Piece Placement #2			/10	
Piece Placement #3			/10	
Turn Timing			/6	
Game Pause			/6	/32
Piece Placement #4			/4	
Joystick			/5	
Sound Effects			/6	/15

Total: (out of 100)	
General deductions: (errors in the program that do not fall into any above category, e.g general lag in gameplay)	
Penalties: (code compilation, incorrect submission files, etc. Does not include late penalty)	
Final Mark: (excluding any late penalty which will be calculated separately)	