CSSE2010 & CSSE7201

Assignment Two

Semester Two, 2023

Due: 4:00pm, Friday 27th October (AEST)

Weighting: 20% (100 marks)

The University of Queensland

School of Electrical Engineering and Computer Science

Objective

As part of the assessment for this course, you are required to undertake an assignment which will test you against some of the more practical learning objectives of the course, namely your C programming skills applied to the ATmega324A microcontroller.

You are required to modify a program to implement additional features. The program is a basic template of the game "Guitar Hero" (described in detail on page 3). The AVR ATmega324A microcontroller runs the program and receives input from multiple sources and uses an LED matrix as a display for the game, with additional information being output to a serial terminal and – to be implemented as part of this assignment – a seven-segment display and other devices.

The version of Guitar Hero provided to you has very basic functionality – it will present a splash screen upon launch, respond to push button presses or a terminal input "s"/"S" to start the game, then display the board for the game Guitar Hero with the notes and timing area. You can add features such as playing the notes, game scoring, pausing, audio etc. The various features have different tiers of difficulty and will each contribute a certain number of marks. Note that marks are awarded based on demonstrated functionality only, regardless of how much effort or code has gone into attempting such functionality.

Don't Panic!

You have been provided with approximately 1500 lines of code to start with – many of which are comments. Whilst this code may seem confusing, you do <u>not</u> 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 matrix 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. Several intro/getting started videos are available on Blackboard, which contains a demonstration of some of the expected functionality to be implemented and walks through setting the project up with the provided base code, and how to submit. Note that the requirements in this document takes priority over anything shown in the feature demonstration videos.

Grading Note

As described in the course profile, if you do <u>not</u> score at least 10% on this assignment (before any late penalty) then your course grade will be capped at a 3 (i.e., you will fail the course). If you do <u>not</u> obtain at least 50% on this assignment (before any late penalty), then your course grade will be capped at a 5. Your assignment mark (after any late penalty) will count 20% towards your final course grade.

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 any 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.

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 game event loop and examples of how time-based events are implemented. You should read and understand this file.
- `game.c/.h` – this file contains the implementation of the game components and is used to store the state of the game. You should read this file and understand what representation is used for the game state and the note representation. You will need to modify this file to add required functionality.
- `display.c/.h` – 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.
- `buttons.c/.h` – this contains the code which deals with the push buttons. It sets up pin change interrupts on those pins and records rising edges (buttons being pushed).
- `ledmatrix.c/.h` – 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` – this file contains definitions of some useful colours for use with the LED matrix.
- `serialio.c/.h` – 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).
- `terminalio.c/.h` – 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.
- `spi.c./h` – 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.

- `timer0.c/.h` – sets up a timer that is used to generate an interrupt every millisecond and update a global time value that is used to time various game events (such as the Guitar Hero note movement). This can be useful for implementing any features that require precise timing.
- `timer1.c/.h` & `timer2.c/.h` – largely empty files, that can be used for features that require timers/counters one and two.

NB: When you create a new project in Microchip Studio, a `main.c` file will automatically be created, containing an empty `main()` function. `project.c` also contains a `main()` function, but Microchip Studio will preferentially look the `main()` function in the `main.c` file, if it exists. Please ensure that you delete the `main.c` file so that Microchip Studio will look for the `main()` function in `project.c`.

Guitar Hero Description

This assignment involves creating a replica of the classic game "Guitar Hero". Guitar Hero is a one-player game that coarsely simulates playing a guitar, and is in a broader genre of rhythm games that also includes Rock Band and Dance Dance Revolution. There are four "lanes", each two columns wide. Musical notes, in red, will come in from the top of the screen, and move down one row after a fixed duration. When they are in the scoring area, in yellow, the player should press a push button to play that note. The notes in the rightmost lane are played with B0, then B1, B2, and finally the notes in the leftmost lane are played with B3. The bright yellow row is the perfect timing for each note, while the darker yellow areas are less precise.

When a note is played (by pressing an IO board push button or serial terminal key, see Tier A features below), it should turn green. Notes can only be played in the scoring area (i.e., the yellow regions). The top row of the lane that the next note will appear in is coloured dark red.

A diagram of the LED matrix is shown in Figure 1 at the top of the next page. It contains information relevant to the initial operation, as well as the **Play Notes with Push Buttons**, **Play Notes with Terminal Inputs**, **Terminal Game Score**, and **Audio** features.

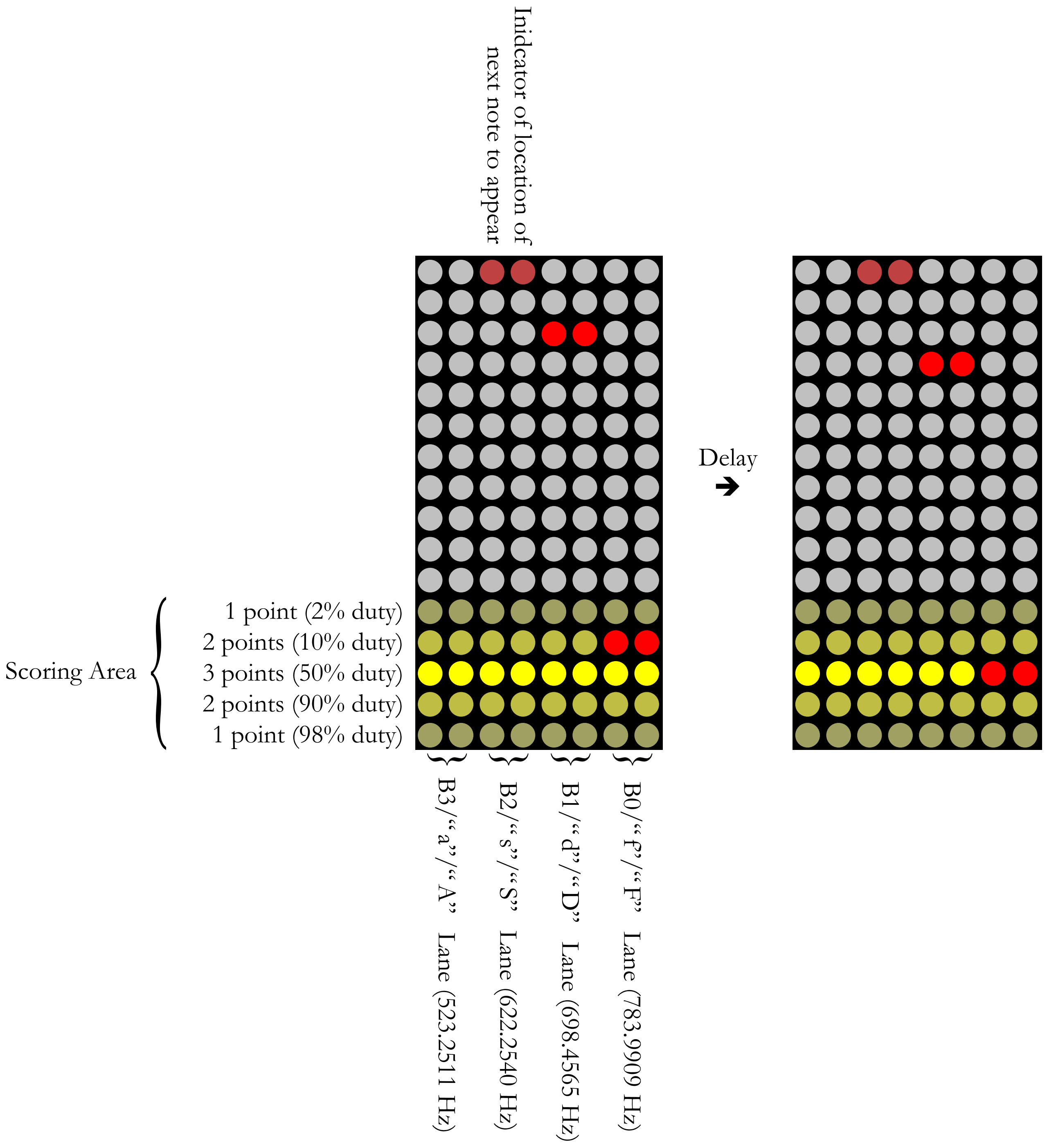
Initial Operation

The provided program has very limited functionality. It will display a splash screen which detects the rising edge on the push buttons B0, B1, B2 and B3, as well as the input terminal character "s"/"S". Pressing any of these will start a game of Guitar Hero.

Once started, the provided program detects a rising edge on the push button B0, but no action is taken on this input (this will need to be implemented as part of the **Play Notes with Push Buttons** feature).

Wiring Advice

When completing this assignment, you will need to make additional connections to the ATmega324A microcontroller to implement particular features. To do this, you will need to choose which pins to make these connections to. 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 (included at the end of this document. A standalone version is also available on Blackboard). Hint: Before implementing any features, read through them all, and consider what peripherals each feature requires and any pin limitations this imposes. If you do not do this, you may find yourself in a situation where the pins that must be used for a peripheral for a later feature are already connected to another peripheral, requiring you to rewire your breadboard and update your code before you can use the new peripheral. Some connections are defined for you in the provided base code and are shown in grey in the table on the next page.



Inidcator of location of next note to appear
Delay
Scoring Area
1 point (2% duty)
2 points (10% duty)
3 points (50% duty)
2 points (90% duty)
1 point (98% duty)
B3/"a"/"A" Lane (523.2511 Hz)
B2/"s"/"S" Lane (622.2540 Hz)
B1/"d"/"D" Lane (698.4565 Hz)
B0/"f"/"F" Lane (783.9909 Hz)

Figure 1: LED Matrix Diagram for Guitar Hero

Wiring Table

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 B3	Button B2	Button B1	Button B0
C								
D							Serial RX Baud rate: 19200	Serial TX

Program Features

Marks will be awarded for features as described below. Part marks will be awarded if part of the specified functionality is demonstrated. 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. You may implement higher-tier features without implementing all lower-tier features if you like (subject to prerequisite requirements), though it is highly encouraged to implement the **Manual Mode** feature as soon as practical. 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%, and marks may be deducted if features interact negatively with one another, and the number of potential iterations grows quadratically with the number of features implemented.

You may modify any of the code provided (unless indicated otherwise) 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 may tell you which code to modify or there may be comments in the supplied code to guide you.

Note: The course has a pass hurdle of 10% for this assignment, which can be achieved by completing the first two features (**Splash Screen** and **Play Notes with Push Buttons**). See Grading Note for further details.

Minimum Performance — Tier X: 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 splash screen, and display the initial game when a push button or "s"/"S" is pressed. No marks can be earned for other features unless this requirement is met, i.e., your assignment two mark will be zero.

Splash Screen — Tier A: 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, in the indicated location (remove the placeholder text, including the chevrons `<>`). Do this by modifying the function `start_screen()` in file `project.c`.

The Terminal Summary section shows all features that require printing text to the terminal. You may find it useful to read through this and plan your terminal layout to avoid running out of room or having messages collide with each other.

Play Notes with Push Buttons — Tier A: 8 marks

The provided program does not allow playing notes. Modify the program so when push button B0 (connected to pin B0) is pressed, if there is a note within the scoring area (i.e., the yellow regions) of the rightmost lane, that note will turn green, and remain green as it moves down until it moves off the screen. Similarly, pressing push button B1 (connected to pin B1) should play any note in the next lane in the scoring area, push button B2 (connected to pin B2) should play any note in the next lane in the scoring area, and push button B3 (connected to pin B3) should play any note in the leftmost land in the scoring area. These push button assignments are shown in Figure 1. The process of played notes being marked green, and remaining green, is demonstrated in Figure 2.

If there is no note in the scoring area of the corresponding lane, nothing should happen at this stage (this will change when the **Terminal Game Score** feature is implemented).

The playing should occur once on each push button press. If a note enters the scoring area while the corresponding push button is being held down, nothing should happen; the push button will need to be released and repressed to play that note.

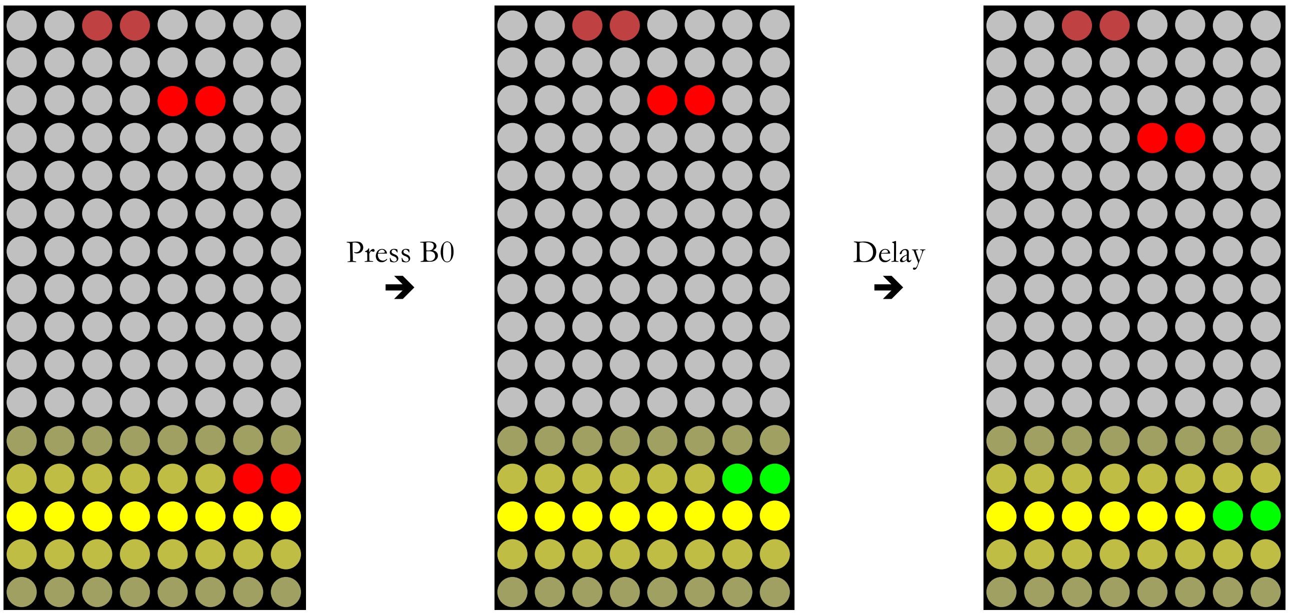


Figure 2: Behaviour of played notes

Hints: In the `play_game()` function in the file `project.c`, when push button B0 is pressed, the function `play_note(0)` in the file `game.c` is called. This function is currently empty; start by filling in the `play_note()` function (there are some hints to get you started).

NB: Playing a note for this project does <u>not</u> mean sounding a tone, and so audio does <u>not</u> need to be implemented for this feature. However, during the **Audio** feature later, you will implement sounding a tone when a note is played.

Play Notes with Terminal Input — Tier A: 4 marks

The provided program does <u>not</u> register any terminal inputs once the game has started. Modify the program such that pressing "a"/"A" should play a note in the leftmost lane, "s"/"S" and "d"/"D" in the next two lanes, and "f"/"F" in the rightmost lane (all in the scoring area), in a similar manner to the previous task. These key assignments are shown in Figure 1. Note that both the lower case and upper case of each letter should execute these movements as described. Also note that the inbuilt serial functionality handles keyboard inputs that are held down for you.

On the splash screen, the game can be started by pressing "s"/"S"; looking at the function `start_screen()` should give you an idea of how to read serial input from the terminal. Do <u>not</u> make other keys start the game from the splash screen.

Terminal Game Score — Tier A: 8 marks

*Requires **Play Notes with Push Buttons** and/or **Play Notes with Terminal Input** features to be implemented.*

When the player plays a note, award them points, depending on the timing:

- If the note is in the centre row of the scoring area, award them three points;
- If the note is in the second or fourth row of the scoring area, award them two points;
- If the note is in outside rows of the scoring area, award them one point.

These values are shown in Figure 1.

If the player tries to play a note when there is no note in the corresponding lane within the scoring area, deduct one point. This includes playing a note that has already been played (i.e., a note that has already turned green). If a note slides off the end of the track without being scored (i.e., still red), deduct one point.

On the terminal, display the score, and update it whenever a point is scored. Serial data for the score should only be sent when the score changes; the game should not spam the terminal constantly. Align the score such that digits of the same magnitude always appear in the same location, regardless of how large the score is. I.e., the rightmost digit of the score should always appear in the same location, regardless of if the score is one, two, or three digits, and regardless of if the score is negative. Do not use leading zeroes to pad the score. For example, the following would all be compatible (using "·" to represent a space for visibility):

```
Game·Score:····1              Game·Score:··123              Game·Score:··-12
Game·Score:···12              Game·Score:···-1              Game·Score:·-123
```

The score should not be displayed on the terminal during the splash screen.

Game Over — Tier A: 4 marks

Once the last note in the track has slid off the bottom of the LED matrix, the game should end. The terminal should display an end screen that indicates this, which must be notably different from the splash screen and screen that is displayed while the game is running. When "s"/"S" or any IO board push button is pressed, the game should return to the splash screen (for both the LED matrix and terminal). There may be a delay between the last note sliding off and the end screen being shown, so long as such a delay is not unreasonably long.

Later features, namely the **Game Speed** and **Custom Track** features, allow for game configuring on the splash screen. These features should not be configurable on the end screen. The **High Score** feature will modify the end screen. The **Seven-Segment Game Score** feature also distinguishes between the splash screen and the end screen. The Terminal Summary section shows all features that require printing text to the terminal during the end screen.

Manual Mode — Tier A: 8 marks

This feature is designed to assist with testing most of the other features. You are highly encouraged to implement this feature before implementing any later features.

When "m"/"M" is pressed during the game, the notes should stop moving. Until "m"/"M" is pressed again, the game is in manual mode. While in this mode, pressing "n"/"N" will move the notes one position. It should be possible to enter manual mode while a game is running, not just from the splash screen. The terminal should show a message when the game is in manual mode, and clear the message upon exit from manual mode.

Unlike the **Game Pause** feature (below), pressing the push buttons or keys should still play notes.

Game Countdown — Tier A: 8 marks

Before a new game starts (after the splash screen), write "3", "2", "1", "GO" on the LED matrix. You must design your own pixel typeface (or find one). The text should be oriented so that they can be read when the scoring zone is positioned at the bottom of the matrix. Each of the four texts should be displayed for the length of two playable notes, as per the game speed (i.e., for 2000ms at this stage; this will be variable when the **Game Speed** feature is implemented). No notes (or dark red upcoming notes) nor the scoring area should appear on the matrix until after "GO" has been removed.

Seven-Segment Display Game Score — Tier A: 8 marks

*Requires **Terminal Game Score** feature to be implemented.*

Display the score on the seven-segment display. The seven-segment display cannot show the score on the splash screen. Once the game starts, the seven-segment display should show the score throughout the game and on the end screen.

When the player's score is between 0 and 9, inclusive, the left digit should be blank.

When a player's score is negative, "−" (the `G` segment) should be displayed on the left digit (and the score value on the right digit). If a player's score reaches −10 or less, both digits should display "−".

If a player's score reaches or exceeds 100, the rightmost two digits should be displayed, including a leading zero for scores between 100 and 109, inclusive (and between 200 and 209 and so on). The terminal should still display the complete score for scores above 100.

The seven-segment display should always reflect the current score, and no display flickering or "ghosting" (where digits can be faintly seen on the opposite display) should be apparent. Both digits should be of equal brightness. Include any connections required on your submitted feature summary form.

Game Speed — Tier B: 6 marks

Add the ability to toggle between different game speeds using the terminal. There should be three different game speeds:

- Normal speed should have one playable note per 1000 milliseconds and is the default game speed, already implemented;
- Fast speed should have one playable note per 500 milliseconds; and
- Extreme speed should have one playable note per 250 milliseconds.

This speed refers to the note-to-note duration. The notes should move down the lanes at five times this speed.

The game speed may be changed when on the splash screen by inputting keys with the terminal:

- Inputting "1" selects normal speed;
- Inputting "2" selects fast speed;
- Inputting "3" selects extreme speed.

The game speed should <u>not</u> be changeable during the game. You may also choose to retain the previous game speed when a new game starts, but this is also <u>not</u> a requirement. As the user changes the game speed, the splash screen animation on the LED matrix should also change speed.

The current game speed should be shown on the terminal as soon as the device is powered, and appropriate text should be added to the terminal display to indicate what the game speed is at all times. As is the case for the game score, the current game speed text should only update when the game speed changes. The text should be more descriptive than "speed 1", "speed 2", "speed 3" or similar.

Game Pause — Tier B: 6 marks

Modify the program so that if the "p"/"P" key on the serial terminal is pressed then the game will pause. When "p" or "P" key on the serial terminal is pressed again, the game recommences. All note playing push buttons/key presses should be discarded whilst the game is paused; you will need to handle the button queue while the game is paused. You may wish to allow game features (such as entering manual mode) to function while the game is paused, but this is <u>not</u> required. The note update timing must be unaffected – e.g., if the pause happens 150ms before the note position is updated, when at normal speed (i.e., 200ms per movement/1000ms per playable note), then the note position should be updated 50ms after the game is resumed, <u>not</u> immediately upon resuming. Other functionality such as the seven-segment display should be unaffected i.e., both digits of the seven-segment display should still be shown (without ghosting etc.) if implemented.

Use LED L7 to indicate if the program is currently paused. If the game is paused LED L7 should be on, and if the game is <u>not</u> paused LED L7 should be off. Include any connections required on your submitted feature summary form. Also, display the message "Game Paused" on the serial terminal when the game is paused. The message should only be shown while the game is paused, so when the game is unpaused, the pause message should be erased from the terminal. The existing messages shown on the terminal (e.g., game speed, game score) should remain while the game is paused.

Audio — Tier B: 6 marks

Requires ***Play Notes with Push Buttons*** *and/or* ***Play Notes with Terminal Input*** *features to be implemented.*

Add audio to the program which are to be output using the piezo buzzer. These should be tones sounded whenever the player plays a note (by pressing a push button or key). The frequencies should be (as near as practicable) as follows:

- B0 note: 783.9909Hz (G5);
- B1 note: 698.4565Hz (F5);
- B2 note: 622.254Hz (D♯5); and
- B3 note: 523.2511Hz (C5).

In addition, the duty cycle of each tone should reflect the accuracy of the played note:

- If the note is played when in the first row of the scoring area, the duty cycle should be 2%;
- If in the second row, the duty cycle should be 10%;
- If in the third/middle row, 50%;
- Fourth row, 90%; and
- Fifth/last row, 98%.

Both the frequencies and duty cycles are shown in Figure 1.

Each tone should play for the time it takes for the notes on the screen to advance five times, or until a new note would be played. This will ordinarily be for the length of the game speed (e.g., play each tone for 1000ms at normal speed), but while in manual mode, this will instead be until “n”/“N” is pressed five times. You should account for entering/exiting manual mode while notes are playing.

If a player attempts to play a note when there is no note in the corresponding lane, no sound should be produced, and if a tone is currently being played, that tone should stop playing.

Pausing while a tone is playing should mute that tone until the game is unpaused, at which point the tone should resume.

Combo Scoring — Tier B: 6 marks

Requires ***Terminal Game Score*** *feature to be implemented.*

When a player plays a note that is in the centre (three point) row of the scoring area, they should increase their combo count by one. The combo starts the game at zero. Whenever they play a note in a different row, or play a note when there is no note in the corresponding lane within the scoring area, or a note slides off the end of the track, they should reset their combo count to zero. I.e., whenever the score changes, the combo count should also change. Use LEDs L0, L1 and L2 to show the combo count, using a unary system up to three i.e., light up L0 when the combo count is one, light up L0 and L1 when the combo count is two, and light up L0, L1 and L2 when the combo count is three or more (and all LEDs off when the combo count is zero). The true combo count (which can go above three) should be displayed on the terminal while the game is running.

When the combo count reaches three, all unscored notes on the LED matrix should change colour to orange. You will need to define a colour for this, as well as a dark orange for the location of the next note to appear at the top of the matrix. In addition, the player scores four points instead of three whenever they play a note in the centre row of the scoring area. When the combo count resets, the notes should change back to red, and scoring resets to the original method i.e., the player scores three points for notes in the centre of the scoring area. Finally, "Combo!" should be displayed on the serial, in an ASCII art typeface, while the combo count is three. You may use external tools to generate the ASCII art. The ASCII art should consist of at least 300 individual characters. Partial marks may be awarded for "Combo!" (as plaintext rather than ASCII art) being printed. When the combo count is resets, or the game ends, the ASCII art should be removed from the terminal. Printing and removing this text should not cause the game to lag.

Long Notes — Tier B: 6 marks

Requires ***Terminal Game Score*** *and* ***Manual Mode*** *feature to be implemented.*

Implement long notes for the track. There is already data for the long notes encoded in the upper nybbles of each element of `track`. These should be displayed over multiple rows on the LED matrix. When using the push buttons to play the notes, they should be scored as normal for the first pixel. For the subsequent pixels, if the same push button is still held down when the notes move, the next pixel should be scored, for one point (regardless of position in scoring area). Figure 3 below shows an example of scoring this way. Once the push button is released, the remainder of the note cannot be scored; if the push button is repressed, it should count as a mistake, and subtract one point from the score. Other than the first pixel of the note, unscored pixels do not deduct points when the slide off the screen. This feature needs only to be implemented for the push buttons; it is not required (nor recommended) to implement this for the serial input.

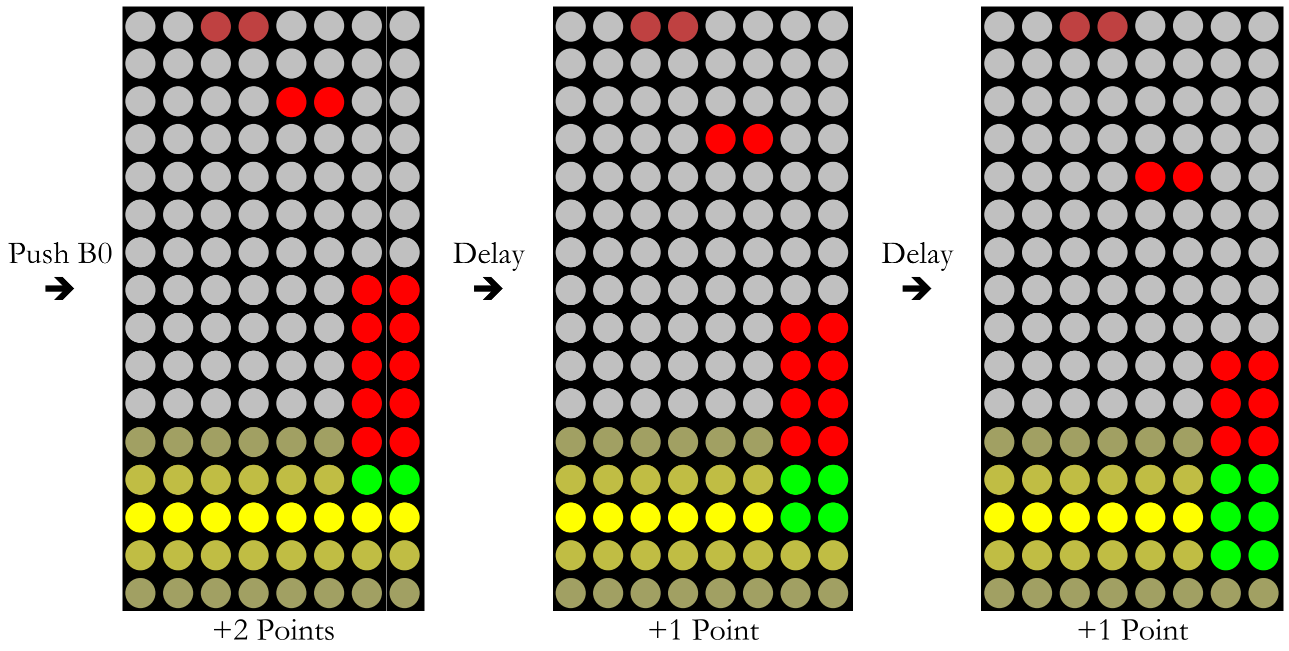


Figure 3: Implementation of long notes

If the **Audio** feature has been implemented, then long notes should play their tones for longer.

Joystick — Tier C: 6 marks

Requires ***Audio*** *and* ***Manual Mode*** *feature to be implemented.*

Use one axis of the joystick to alter the pitch of the audio tones. When the joystick is in the centre position, the pitch of the tones should be the normal pitch as specified in the **Audio** feature. When the joystick is in the far-left position, the pitch of the tones should be two semitones lower than the normal pitch for that note. When the joystick is in the far-right position, the pitch of the tones should be two semitones higher than the normal pitch for that note. Between these prescribed positions, the pitch of the tones should vary in a smooth manner. If two tones are a semitone apart, the ratio of the frequencies will be $1:\sqrt[12]{2} \approx 1:1.05946309436$. As the Atmega324A does <u>not</u> have a floating-point unit, you are permitted to approximate this with a set of piecewise linear functions. For each of these linear functions, you will need to use integer fractions in place of decimals, in the form of $z = x \cdot (ay + c) \div b$, where x is the base frequency for the note's tone, y is the ADC reading, and z is the modified frequency, for some integers a, b and c. The pitch of the tones should reflect the current position of the joystick at all times; rapidly moving the joystick while a tone is playing should produce a vibrato effect. The PWM duty cycles should remain consistent with the requirements in the **Audio** feature when the tone pitch changes. Note that this feature uses only one axis of the joystick; the other axis may be left disconnected.

Custom Track Tier C: 6 marks

Add two more tracks, in addition to the provided track. Give each track a title, including the default track. While on the title screen, the name of the first/default track should be displayed. Pressing "t"/"T" will select the next track, and pressing "t"/"T" again will select the third track, and so on. When "t"/"T" is pressed when the final track is selected, the first track should be selected again.

Each melody should use all four of the lanes, and have at least 48 notes in its duration. If you have implemented the **Long Notes** feature, some of the notes for your custom tracks must also be long notes.

You may choose different frequencies to play on the piezo buzzer for each track if the **Audio** feature is implemented. You may add a fourth and beyond track, but this will <u>not</u> award additional marks, and may interfere with the **High Score** feature.

High Score Tier C: 8 marks

*Requires **Terminal Game Score** feature to be implemented, and both the **Game Speed** and **Custom Track** features must be implemented for full marks.*

NB: This feature goes beyond the content that is taught in the labs. You will be expected to gain the necessary knowledge to complete this feature yourself by consulting the datasheet and relevant references online. Tutors will only provide limited assistance.

When a track is complete, show the high scores for that track and that difficulty on the terminal. Pre-populate this with made-up scores from made-up players (these scores should be beatable, and cover a large range of scores).

If a player beats a score on the scoreboard, display the scoreboard with their score in the appropriate position, and a blank for the name. The player can then type in their name, pressing enter to submit, and the new name and score should be displayed in a different colour. Use the EEPROM to store the high scores, so that the player's new high score will be displayed at the end of future games, even after the device is reset (in the default colour for future games). Ten scores and names should be stored for each track and difficulty combination; when a score is beaten, remove the lowest score. Do <u>not</u> store the high score if the player submits a blank name. If both of the **Game Speed** and **Custom Track** features have been implemented, then each high score should be able to score a name that is up to ten characters long, and contain any printable ASCII character (you do <u>not</u> need to consider tab characters). It is worth noting that if **Terminal Game Score**, **Combo Scoring**, and **Long Notes** have all been implemented, then scores in the four hundreds should be possible with default track, and negative scores can be arbitrarily low.

If the player enters the sequence "r" → "R" → "r" while the end screen is being displayed (and after the player has entered their name, if applicable), the EEPROM should be reset to the pre-populated names and scores.

Terminal Summary

If all features are implemented, then during the splash screen, the terminal should display:

- The "AVR Hero" title – **Splash Screen**;
- Your name and student number – **Splash Screen**;
- A manual mode message while manual mode is active – **Manual Mode**;
- The selected game speed – **Game Speed**; and
- The selected track – **Custom Tracks**.

During the game, the terminal should display:

- The current score – **Terminal Game Score**;
- A manual mode message while manual mode is active – **Manual Mode**;
- The selected game speed – **Game Speed**;
- A pause message while the game is paused – **Game Pause**;
- The current combo count – **Combo Scoring**;
- The Combo ASCII art while the combo count is three or more – **Combo Scoring**; and
- The selected track – **Custom Tracks**.

During the end screen, the terminal should display:

- The final score – **Terminal Game Score**;
- The selected game speed – **Game Speed**;
- The selected track – **Custom Tracks**; and
- The high score table – **High Score**.

You may find this summary useful when planning out your terminal layout.

All information displayed on the terminal should be identifiable as to what it represents. For example, the game score and combo count should have text to identify them, and not just be raw numbers, which could cause them to be confused with one another.

Assessment of Feature Implementation

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 audio should pause.

Features are shown grouped in their tiers of approximate difficulty (Tier A, Tier B, and Tier C).

Submission Details

The due date for the assignment is 4:00pm Friday 27th October. The assignment 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 have not changed them);
- Your final `.hex` file (suitable for downloading to the ATmega324A AVR microcontroller program memory, see the getting started video to locate the `.hex` file); and
- A `.pdf` feature summary form (see below).

Please name your .hex and .pdf in the form assi2_4nnnnnnn.hex/.pdf, where "4nnnnnnn" is your eight-digit student number.

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 only the .hex file is submitted with no source files, then your submission will not be marked. The .hex file will be used if the marker is unable to compile/build the submitted source files.

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.

It is the responsibility of the student to ensure that their submission is correctly uploaded to the Blackboard submission portal with all of the files they intend to submit. This can be ensured by downloading your .zip file after submission is made, un-zipping the submission to check all files are correctly included and checking whether you are able to compile the project successfully. It is suggested that you use the Axon lab computers for this check.

The feature summary forms are on the last pages of this document. A separate electronically fillable .pdf form will be provided on Blackboard. This form can be used to specify which features you have implemented and how to connect the ATmega324A to other devices 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 the .pdf file with your other files. If you have any assumptions or comments to convey to the marker, include these on the feature summary form.

Incomplete or Invalid Code

If your submission is missing files (e.g., will not 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 no changes you made to the missing files will be considered in marking.

If your submission does not compile and/or link in Microchip Studio version 7 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. This will apply based off of compiling and linking with Microchip Studio on the Axon lab computers, regardless of the results of compiling or linking with other programs. 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 assignment (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, then a mark deduction will apply – 1 mark penalty per warning up to a maximum of 10 marks. The warning list generated by Microchip Studio on the Axon lab computers from a clean build will be the canonical list for these penalties. 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 assignment 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. The number of marks deducted will depend on the severity of the issues.

Late Submissions

As per the course profile, where an assessment item is submitted after the deadline (i.e., 4:00pm Friday 27th October), without an approved extension, a late penalty will apply. Assessment submissions received after the due time (or any approved extended deadline) will be subject to a 100% late penalty. A one-hour grace period will be applied to the due time after which time the 100% late penalty will be imposed.

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 the My Grades section of Blackboard.

CSSE7201

The AVR programming described in this document constitutes part two of the CSSE7201 assessment. Please see Blackboard for part one, and/or the course profile for more information regarding the assessment details.

Version

This is version four of the assignment two document. Changes between the current version and the previous version are shown in green. Changes between the previous version and version one are shown in blue.

The University of Queensland – School of Electrical Engineering and Computer Science
Semester Two, 2023 – CSSE2010/CSSE7201 Assignment Two – Feature Summary

Student Number								Family Name	Given Names
4									

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 have 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 B3	Button B2	Button B1	Button B0
C								
D							Serial RX	Serial TX
							Baud rate: 19200	

Feature	✓ if attempted	Comment (Anything you want the marker to consider or know?)	Mark	
Splash Screen			/4	
Play Notes with Push Buttons			/8	
Play Notes with Terminal Input			/4	
Terminal Game Score			/8	
Game Over			/4	
Manual Mode			/8	
Game Countdown			/8	
Seven-Segment Display Game Score			/6	/50
Game Speed			/6	
Game Pause			/6	
Audio			/6	
Combo Scoring			/6	
Long Notes			/6	/30
Joystick			/6	
Custom Tracks			/6	
High Score			/8	/20

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)	