CAB202 Microprocessors & Digital Systems

Assessment 2: Microcontroller Project

Task Summary

Your task is to implement the Simon game on the QUTy. The classic Simon game is shown in **Figure 1**.

Figure 1: Simon game[1].

This game produces a sequence of lights and tones which the user is required to reproduce. If the user succeeds, the sequence becomes progressively longer.

On the QUTy, the four pushbuttons will perform the function of the four coloured buttons in the Simon game. Each pushbutton is mapped to the vertical segments directly above it on the 7-segment display and a unique tone as per **Figure 2**. For each press of a pushbutton, the corresponding segments will be illuminated, and a tone will sound, as specified in **Table 1**.

[1] https://en.wikipedia.org/wiki/Simon_(game)

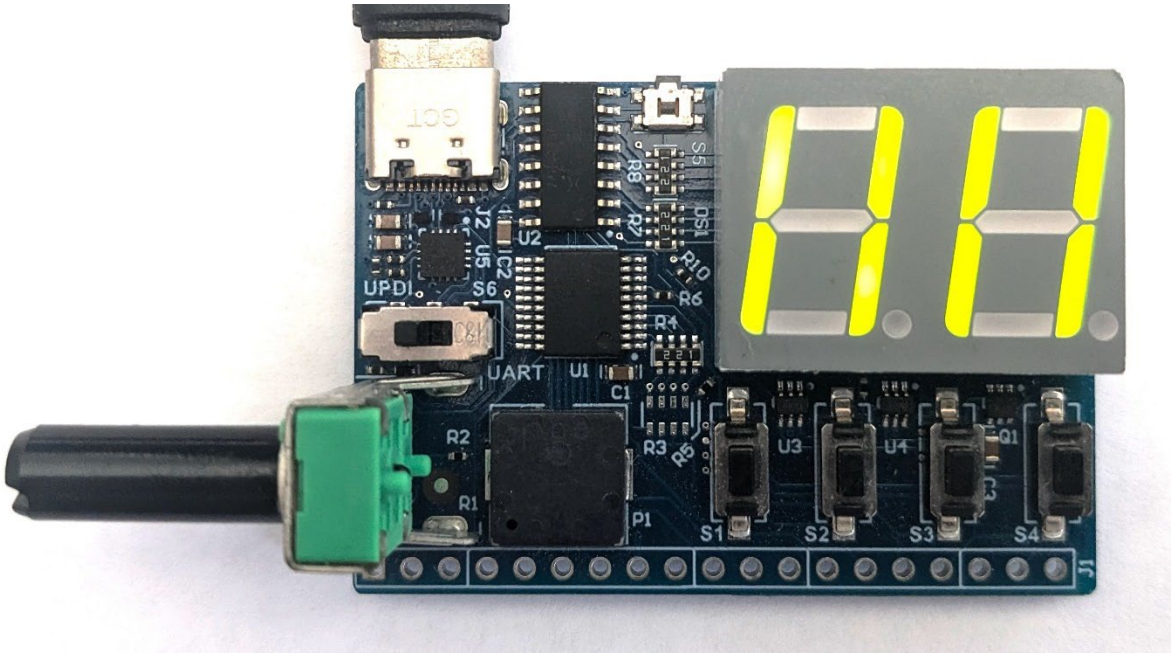
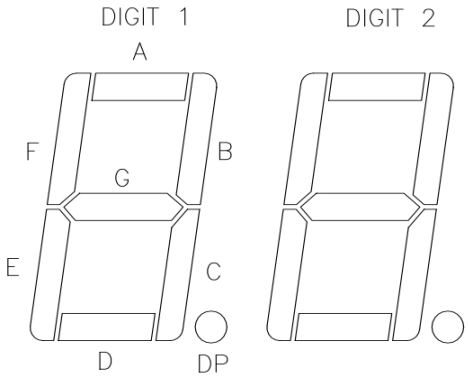


Figure 2: Pushbuttons and corresponding LED segments.



(a)

Pushbutton	Segments	Tone
S1	Digit 1: E, F	E(high)
S2	Digit 1: B, C	C♯
S3	Digit 2: E, F	A
S4	Digit 2: B, C	E(low)

(b)

Table 1:
(a) LED segment label mappings.
(b) Pushbuttons and their corresponding LED segments and tones.

The frequencies of the tones mapped to each pushbutton will depend on your student number. More information is provided in **Table 2**.

Functionality

Your program is required to implement the following core functionality:

Section A. Gameplay

- Upon system reset, or at the start of a new game, the **sequence length** is set to **one**. The sequence length corresponds to the user's **score**.
- Upon system reset, and on each turn of gameplay, "Simon" begins a level by producing a tone on the buzzer and illuminating two segments of the 7-segment display (see **Table 1**). Simon will play a tone and illuminate two segments for all **steps** in the **sequence** determined in Section B. Each **step** is active for 50% of the duration of the **playback delay** (see Section C), after which the buzzer is silent and the 7-segment display is blank, for the remainder of the **playback delay**. Following this, the next **step** in the sequence will be active.
- Once the **sequence length** is reached, the user must press the pushbuttons corresponding to the steps in the same sequence. As each pushbutton is pressed, the corresponding segments of the 7-segment display are illuminated, and the corresponding tone is sounded. This occurs for either: 50% of the duration of the **playback delay**, or the duration of the button press, whichever is longer.
- If the user's sequence matches Simon's, the SUCCESS pattern is displayed for the **playback delay**. Finally, the **sequence length**, which is equal to the user's score, is increased by **one**.
- The next turn replicates the steps from the previous sequence, while appending an additional step at the end.
- The game repeats until the user fails to match Simon's sequence, or when the maximum **sequence length** is reached.
- The game ends as soon as an incorrect input is provided by the user. This causes the FAIL pattern to be displayed for the **playback delay**, followed by the user's score, which is also displayed for the **playback delay**. Following this, the 7-segment display will remain blank for the **playback delay**. After this, the **sequence length** is reset back to **one** and a new game will begin.
- The user's score is displayed as a decimal number between 0 and 99, on the 7-segment display. In the unlikely event that a score exceeds 99, the two least significant digits will be displayed. Leading zeroes will be shown only if the score exceeds 99. For example, if the user's score is 8, ' 8' will be shown, however, if the user's score is 108, '08' will be shown.
- The **SUCCESS pattern** requires **all 7 segments** on both displays to be illuminated.
- The **FAIL pattern** requires the **G segment** on both displays to be illuminated.

Section B. The Sequence

- The **steps** in a **sequence** are generated using a Pseudo-Random Number Generator (PRNG). This PRNG is implemented using a Linear-Feedback Shift Register (LFSR) with the mask: **`MASK = 0xE2023CAB`**. The *initial* state of the LFSR represents the **seed** of the PRNG, which must be initialised with your student number, encoded as a 32-bit hexadecimal literal.
  Example:
  Student number: n12345678 → STATE_LFSR = 0x12345678
  Student number: n5556667 → STATE_LFSR = 0x05556667
- Each **step** is derived from STATE_LFSR through the following algorithm:

```
BIT ← lsbit(STATE_LFSR)
STATE_LFSR ← STATE_LFSR >> 1
if (BIT = 1)
     STATE_LFSR ← STATE_LFSR xor MASK
STEP ← STATE_LFSR and 0b11
```

- The value of STEP specifies the tone to be played and the segments to be illuminated. `STEP = 0` means that E(high) is to be played and segments EF are to be illuminated on DIGIT 1. The user must then press pushbutton S1 to reproduce this step. Similarly, `STEP = 1` means that C♯ is played and segments BC are illuminated on DIGIT 1, and so on.
- Upon successfully reproducing a sequence of steps, the sequence of steps in the next turn will contain the same steps as in the current sequence, with an additional step added to the end.
- If the user fails to match Simon's sequence, the game will restart, and a new sequence will be generated. This new sequence will commence from where the current sequence ended.
  Example:
  Assume the seed `0x12345678` where the user has reached turn 7 with the sequence `1341343` and expected subsequent sequence `13413432`. If the user incorrectly enters a `1` instead of a `3` on the 5th step, the game will end and restart from the 8th step. The sequence in the next turn will then have a length of **one** and consist of the step 2.
- Upon system reset, the **seed** must be set back to its initial value, allowing the Simon game to be replayed.

Section C. Playback Delay

- The **playback delay** defines a duration of time used to control several aspects of the game, as discussed in Section A. It is defined using the position of the potentiometer and ranges from 0.25 to 2.0 seconds.
- The playback delay can be **increased** by turning the potentiometer **clockwise** (direction when facing the potentiometer).
- The playback delay can be **decreased** by turning the potentiometer **anti-clockwise** (direction when facing the potentiometer).
- The playback delay is calculated as a linear interpolation between 0.25 and 2.0 seconds, that depends on the position of the potentiometer. In other words, if the potentiometer is exactly halfway between the most clockwise and most anti-clockwise positions, the playback delay will be exactly halfway between 0.25 and 2.0 seconds.

Section D. Playback Frequency

- The **playback frequencies** of the four tones will default to the values derived from your student number, as shown in **Table 2**.
- The octave of the playback frequencies can be **increased** by receiving the "INC FREQ" function through UART.
- The octave of the playback frequencies can be **decreased** by receiving the "DEC FREQ" function through UART.
- When the above tokens are received, playback frequencies will be incremented or decremented in steps of **one octave** (frequencies are multiplied or divided by 2).
- When the above functions are received, **all four frequencies** will be incremented/decremented at once.
- All frequencies should remain within the range of human hearing (20 Hz to 20 kHz).

The four playback frequencies can be calculated using the following table:

E(high)	C♯	A	E(low)
$4\text{xy} \times 2^{\frac{-5}{12}}$	$4\text{xy} \times 2^{\frac{-8}{12}}$	4xy	$4\text{xy} \times 2^{\frac{-17}{12}}$

Table 2: Playback frequencies for the Simon game.

1. xy represents the last two digits of your student number.
2. All frequencies are in Hz and rounded to the nearest integer.

Example:

If $\text{xy} = 40$, E(high) = 330 Hz, C♯ = 277 Hz, A = 440 Hz, and E(low) = 165 Hz.

Section E. Gameplay through UART

In addition to pushbutton inputs, the user may also transmit several commands via the serial monitor to perform various actions. These keys and their corresponding actions are summarised in **Table 3**.

For the following section, the UART interface must be configured to 9600—8-N-1.

Gameplay, INC/DEC FREQ, RESET, and SEED

- The input from a pushbutton can alternatively be produced from its corresponding key. Upon transmitting one of these keys, the same output must be observed from the buzzer and 7-segment display, where both the buzzer and 7-segment display are active for 50% of the playback delay.
    - When the user successfully matches Simon's sequence, the string "SUCCESS\n" is transmitted via UART, followed by the user's score in decimal (up to 65535) and a newline ("\n"). This should occur while the SUCCESS pattern is being displayed on the 7-segment display.
    - When the user fails to match Simon's sequence, the string "GAME OVER\n" is transmitted via UART, followed by the user's score in decimal (up to 65535) and a newline ("\n"). This should occur while the FAIL pattern is being displayed on the 7-segment display.
      After the score has been displayed for the playback delay, and the display has been blank for the playback delay, the user may be prompted to enter their name, after which the high score table is transmitted via UART (see *High Scores*).
- The **INC FREQ** and **DEC FREQ** functions can be used to **increment** or **decrement** the frequency of the tones discussed in Section D. These keys must be handled immediately if they are transmitted while the buzzer is active. Changes to the frequency are retained until system reset (**RESET**).
- The **RESET** key immediately ends the current game, resets all frequencies to their default values, resets the LFSR to its initial state, and resets the sequence length back to one.
- The **SEED** key must be followed by 8 lowercase hexadecimal digits. These 8 hexadecimal digits are used to overwrite the seed of the PRNG. This new seed must only be applied to the next game, whether it is a result of a win, a loss, or the RESET function. The steps generated in all subsequent sequences must be generated from this new seed. If any of the 8 characters following the SEED key are not lowercase hexadecimal digits, no action should be taken.

Key	Function	Action taken
'1' or 'q'	S1	S1 during gameplay
'2' or 'w'	S2	S2 during gameplay
'3' or 'e'	S3	S3 during gameplay
'4' or 'r'	S4	S4 during gameplay
',' or 'k'	INC FREQ	Increase frequency of tones
'.' or 'l'	DEC FREQ	Decrease frequency of tones
'0' or 'p'	RESET	Reset frequencies to default and sequence index to 0
'9' or 'o'	SEED	Load new seed for pseudo-random sequence

Table 3: ASCII characters received through the UART and actions taken. Each function supports two keys.

- All keystrokes are assumed to be instantaneous, and you should not handle cases where a key is held down continuously.

High Scores

- A high score table is used to store the highest scoring games.
- At the conclusion of a game, if the user's score is within the top 5 highest scores, the prompt "`Enter name: `" is transmitted via UART, and the user is able to enter their name. The user's name can contain a maximum of 20 characters.
    - User input is concluded by transmitting a newline ("\n"), that is, by pressing the Enter key[2].
    - If no input is received within 5 seconds of transmitting the prompt, the name is set to an empty string.
    - If a newline is not received within 5 seconds of the last input, the name is set to the characters received so far.
    - If more than 20 characters are transmitted, the name is set to the first 20 characters transmitted.
- If the user is prompted to enter their name while the payload to the SEED function is being transmitted, the user's name should take precedence over entering the new seed. Once the program has accepted the user's name, the user can resume entering a new seed.

Please see overleaf.

[2] Windows users may choose to ignore the carriage return character ("\r") as a newline is represented by two control characters: "\r\n". See https://en.wikipedia.org/wiki/Newline#Representation.

- When transmitted via UART, each entry in the high score table must be formatted as "`<name> <score>\n`", in descending order of score. If fewer than 5 scores are recorded, only existing entries must be transmitted.
  Example:
  Given four recorded entries. Note that names do not need to be sorted when scores are equal.

```
Alice 20
Bob 18
David 9
Charlie 9
```

- The high score table must be stored in SRAM and will therefore be cleared on system reset. The high score table must not be cleared when the RESET function is received.

Note: Your implementation must not use bit banging (manual driving of the SPI or UART interfaces without explicit use of the associated peripherals). It also cannot use the functions provided in the QUTy I/O or QUTy Serial libraries.
At the time of assignment release, some content relevant to the assignment may not yet be covered, and appropriate implementation methods for some functionality may be revisited later in the semester. All tutorials following the release of this assignment are designed to assist you in completing this assignment.
Serial output must adhere solely to the specification of Section E; all additional debugging output must be suppressed in your final submission.

Deliverables

Source code (submitted to Gradescope) (30%) **11:59pm Friday 31st May**