The University of Queensland
School of Information Technology and Electrical Engineering
Semester 1, 2014

CSSE2010 / CSSE7201

PROJECT

Due: **5pm Friday June 6, 2014**
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 Atmel Studio environment.

You are required to modify a program in order to implement additional features. The program is a version of Frogger – see www.happyhopper.org for a web-based version of Frogger if you are unfamiliar with the game. 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.

The version of Frogger provided to you will implement basic functionality – the frog can only move in one direction (forward) and the game ends immediately when a frog “dies”. You can add features such as scoring, other directions of movement, increasing the speed of play, 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 over 1900 lines of code to start with. 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.

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 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.

Individual or Group of Two

You may complete this project individually or as part of a group of two. You are required to tell us, via the course Blackboard site, by **12 noon Thursday May 29, 2014** whether you are completing the project individually or as part of a group of two students. If you are completing it in a group, you must tell us who your partner is and they must also enter your details via the course Blackboard site. Failure to complete this form (by both partners) means that you will be assumed to be completing this project individually.

A group of two will be required to do additional work to get the same mark as an individual; however the amount of work is less than twice that required of an individual. Both members of a group will receive the same mark for the project. Certain features are intended mainly for groups, i.e., groups will need to implement these features while for individuals they are optional and worth fewer marks.

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.c** – this file contains the implementation of the operations (e.g. movements) in the game. You should read this file and understand what representation is used for the game. You will need to modify this file to allow the frog to move in other directions.
- **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.c** – 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
- **score.c** – a module for keeping track of and adding to the score. This module is not used in the provided code.
- **scrolling_char_display.c** – this contains code which provides a scrolling message display on the LED matrix board.
- **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.c** – this file encapsulates all SPI communication. Note that by default, all SPI communication uses busy waiting – the “send” routine returns only when the data is sent.
- **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.c** – sets up a timer that is used to generate an interrupt every millisecond and update a global time value.

Initial Operation

The provided program responds to the following inputs:

- Rising edge on the button connected to pin B2
- Serial input charcter ‘u’ or ‘U’
- Serial input escape sequence corresponding to the cursor-up key

All of these move the frog forward (up) by one row.

Code is present to detect the following, but no actions are taken on these inputs:

- Rising edge on buttons connected to B3, B1 and B0 (intended to be left move, down move, and right move respectively).
- Serial input characters ‘d’, ‘D’, ‘l’ (lower-case L), ‘L’, ‘r’, ‘R’ (intended to be move down, move down, move left, move left, move right and move right respectively)
- Serial input escape sequences corresponding to the cursor left, cursor down and cursor right keys.
- Serial input characters ‘p’ and ‘P’(intended to be the pause/unpause key)

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 and allow the frog to be moved forward. No marks can be earned for other features unless this requirement is met, i.e., your project mark will be zero.

Splash Screen (Level 1)

Modify the program so that when it starts (i.e. the AVR microcontroller is reset) it scrolls a message to the LED display that includes the student number(s) of the student(s) whose project this is. You should also change the message output to the serial terminal to include your name(s). Do this by modifying the function `splash_screen()` in file *project.c*.

Scoring #1 (Level 1)

Add some method of keeping and displaying the score to the program. It is suggested that every time a frog reaches the other side, the score is increased. You should make use of the function `add_to_score(int value)` declared in *score.h*. You should call this function (with an appropriate argument) from any other function where you want to increase the score. If a .c file does not already include *score.h*, you may need to #include it. You will also need to add code to display the score (to the serial terminal in a fixed position) when it changes. If you prefer, you can use the seven segment display for the score (if it is not being used for the “time limit” feature – see below).

Move Frog Left, Right, Down (Level 1)

The provided program will only move the frog up. You must complete the `move_frog_left()`, `move_frog_right()` and `move_frog_backward()` functions in file *game.c*. These move functions should not allow the frog to jump off the game area. The frog position should be updated and redrawn. It should be checked whether the frog will be alive or not. (The frog will die if it jumps into a vehicle or jumps into the river.)

Multiple Lives (Level 1)

Allow the player to have multiple “lives” (chances), e.g. 3 frogs must die before the game is over. Indicate the number of lives remaining by using individual LEDs on the IO board (e.g. LD0 to LD2).

Scrolling Speeds (Level 1)

The provided program scrolls traffic lanes and logs one pixel per second. Modify the program so that each of the five lanes/channels scrolls at slightly different speeds. Speeds should initially range between one pixel per 600ms to one pixel per 1.2s.

New Game (Level 1 – group feature)

Add a feature so that if the ‘n’ or ‘N’ key is pressed on the serial terminal, a new game is started (at any time, even if the current game is in progress). All aspects of the game should be reset to the starting point.

Random positions (Level 1 – group feature)

The provided program always places a new frog in the middle of the bottom row and the vehicle/log patterns always start in the same location. Modify the program so that it places a new frog in a random column (still in the bottom row) and starts each traffic lane/river channel in a random position. You will need to add appropriate code to the functions `put_frog_at_start()` and `init_game()` in the file *game.c*. (Random number seeding is not expected – positions may be the same after each reset – they should just be different from game to game or level to level.)

High Score (Level 1 – group feature)

Keep track of and display (via the terminal) the high score across several games. (This implies your program can play the game multiple times without having been reset – see the “New Game” description above.) You need only store the high score in RAM (i.e. it will be reset when the microcontroller is reset) – you do not need to store the score persistently (EEPROM). (Groups will also earn marks for this feature if “EEPROM Leader Board” is implemented. Individuals will only earn marks for this feature if “EEPROM Leader Board” is not implemented.)

Game Pause (Level 2)

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 should be discarded whilst the game is paused.) The check for this key press is implemented in the supplied code, but does nothing.

Game Levels (Level 2)

The provided game finishes if all five “holes” on the far riverbank are filled with frogs. Modify the program so that the game will continue if this happens. The LED matrix should quickly scroll the message “Level n” (where n is the number of the next level) as soon as the fifth frog reaches home. When the message has finished, play of the next level starts. (This feature does not require the higher levels to be any different – they can be the same.) If the “multiple lives” feature has been implemented then successful completion of a level should restore a life (up to some maximum, e.g. 5).

Acceleration (Level 2)

Make the game speed up as the score gets higher. A suggested approach is that every time a level is completed (and cleared) the game speeds up *slightly*, i.e. the vehicles/logs scroll slightly faster. (Do not speed up play too quickly. An average player should be able to play for at least one minute, but the speed-up must be noticeable within one minute.)

Time Limit (Level 2)

Implement a time limit for each frog (e.g. 12 seconds, though the time may depend on the difficulty of play in your game). If the frog does not make it to the other side before the time runs out, then the life is lost. Use the seven-segment display to show the number of seconds remaining. (The time limit must be greater than 10 seconds so that the use of two digits is demonstrated. For single digit numbers there should be no leading 0 – show a blank.)

Different Game Levels (Level 2 – group feature)

(Prerequisite: Game Levels)
Different game levels should use different vehicle/log patterns and/or colours and/or directions of movement. At least three different patterns should be evident. (The game must be playable in order to demonstrate this within about 60 seconds.)

Random Number Seeding (Level 2 – group feature)

Add a method for seeding the random number generator so that the game is not always the same after reset or power-on. (See the `srand()` function in `stdlib.h`) Seeding is often done based on a timer value – e.g. wait for a user to press a button to start the game and use the time that happens as the seed value.

Auto-repeat (Level 2 – group feature)

Add auto-repeat support to the push buttons which move the frog – i.e., if they are held down then, after a short delay, they should act as if they are being repeatedly pressed.

EEPROM Storage of High Score Leader Board (Level 3)

Implement storage of a leader board (top 5 scores and associated names or initials) in EEPROM so that values are preserved when the power is off. If a player achieves a top-5 score then they should be prompted (via serial terminal) for their name or initials. The score and name/initials must be stored in EEPROM and must be displayed on the serial terminal on program startup and at each game-over. (Consider the situation of the EEPROM initially containing data other than that written by your program. You may need to use a "signature" value to indicate whether your program has initialized the EEPROM for use.)

Sound Effects (Level 3)

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 three events. For example, choose three of:

- frog moving
- frog successfully reaching the far side
- frog dying
- game start-up
- start/end of level

Do not make the tones too annoying! Switch 7 on the IOboard must be used to toggle sound on and off (1 is on, 0 is off). You must specify which AVR pin this is connected to.

Joystick (Level 3)

Add support to use the joystick to move the frog up, left, right and down. This must include auto-repeat support – 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.

EEPROM Storage of Game (Level 3)

Implement storage of the complete game state in EEPROM. If the "s" or "S" key is sent from the serial terminal then the whole game state should be saved. If the "o" or "O" key (for "Open") is later sent (possibly many power cycles later), then the saved game should be retrieved and play should continue on in that game. Note the comment about "signature" requirements above (EEPROM High Score Leader Board) – it should not be possible to "open" a saved game if none was saved.

Advanced Feature of Your Choice (Level 3)

Feel free to implement other advanced features. The number of marks which may be awarded will vary depending on the judged level of difficulty or creativity involved – up to a maximum of 7 marks.

Assessment of Program Improvements

The program improvements will be worth the number of marks shown on the mark sheet at the end of this document. 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. Note that features you implement must appropriately work together, for example, if you implement game pausing and time limits then the count-down timer must also pause and restart.

Features are shown grouped in their levels of approximate difficulty (level 1, level 2, and level 3). Groups of two students must implement additional functionality to achieve the same marks at each level. Some degree of choice exists at level 3, but the number of marks to be awarded here is capped, i.e., you can't gain more than 19 marks for advanced features even if you successfully add all the suggested advanced features. If an individual implements a group feature then this counts as 1 mark towards level 3.

Submission Details

The due date for the project is **5pm Friday 6 June 2014**. The project must be submitted via Blackboard. You must **electronically submit a single .zip** file containing:

- **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 AVR microcontroller program memory); and
- A PDF feature summary form (see below).

All files must be at the top level within the zip file – do not use folders/directories inside the zip file. A penalty of 5% of your mark will apply if you violate this requirement.

Do not submit (or include in your .zip file) any other files. We do **not** want solution, project, .elf, .lss, .map, etc. files. Do not "double zip" your submission. Your submitted zip file must contain only .c, .h, .hex and .pdf files and not another zip file. A penalty of 5% of your mark will apply if you violate this requirement.

Do not submit .rar or other archive formats – the single file you submit must be a zip format file. A penalty of 5% of your mark will apply if you violate this requirement.

If you make more than one submission, each submission must be complete – the single zip file must contain the feature summary form 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 form is on the last page of this document. 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 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.

For those students working in a pair, only one student should submit the files.

Assessment Process

Your project will be assessed during the revision period (the week beginning Tuesday 10 June 2014). You have the option of being present when this assessment is taking place, but whether you are present or not should not affect your mark (provided you have submitted an accurate feature summary form). Arrangements for the assessment process will be publicised closer to the time.

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 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.

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 5pm Saturday June 7, 2014) will be penalised 20%, less than two days late 30% and so on. (The penalty is a percentage of the mark you earn, not of the total available marks.) Requests for extensions should be made to the course coordinator (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.

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/group under <u>any</u> circumstances. You must not look at or copy code from any other student/group. 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.

Notification of Results

Students will be notified of their results at the time of project marking (if they are present) or later via Blackboard's "My Grades".

The University of Queensland - School of Information Technology and Electrical Engineering
Semester 1, 2014 – CSSE2010 / CSSE7201 Project – Feature Summary

	Student Number								Family Name	Given Names
Student #1										
Student #2 (if group)										

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

Feature (For Groups)	**✓ if attempted**	**Comment** (Anything you want the marker to consider or know?)	**Mark (out of indiv/group)**	
Splash screen			4/3	
Scoring #1			9/6	
Moving L/R/D			15/10	
Multiple Lives			15/10	
Scrolling Speeds			12/8	
New Game			1/6	
Random Positions			1/6	
High Score			1/6	/55
Game Pause			6/4	
Game Levels			6/4	
Acceleration			6/4	
Time Limit			8/5	
Diff Game Levels			1/3	
Random Seeding			1/2	
Auto-repeat			1/4	/26
EEPROM Leaders			6/4	
Sound Effects			6/4	
Joystick			6/4	
EEPROM game			6/4	
Other Advanced			max 7/7	/19 max

Total: (out of 100, max 100) □

Penalties: (code compilation, incorrect submission files, etc. Does not include late penalty) □

Final Mark: (excluding any late penalty which will be calculated separately) □