The University of Queensland
School of Information Technology and Electrical Engineering
Semester 1, 2016

CSSE2010 / CSSE7201
PROJECT

Due: **5pm Friday June 3, 2016**
Weighting: **20% (100 marks)**

Objective

As part of the assessment for this course, you are required to undertake a project that 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 Tetris. (If you are unfamiliar with Tetris, there are numerous websites that describe the game and have versions playable within your web-browser.) The AVR ATmega324A microcontroller runs the program and receives input from a number of sources (e.g. serial port input, push buttons, etc.) and outputs information to various devices (e.g. LED matrix).

The version of Tetris provided to you will implement simple block shifting (blocks can only be shifted to the left), rotation and dropping. You can add features such as scoring, clearing filled in rows, increasing the speed of play, sound effects, block preview 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 2200 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 project.c, game.c and blocks.c. You may need to look at the AVR C Library documentation to understand some of the functions used.

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 a form linked from the course Blackboard site, by **12 noon Thursday May 26, 2016** 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.

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 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/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.

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.

Program Description

The program you will be provided with has several C files that 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 and add files if you wish. 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.

The files you are provided with are:

- **project.c** – this is the main file that contains the event loop and examples of how time-based events (e.g. dropping a block) are implemented. You should read and understand this file.
- **game.h/game.c** – game.c contains the implementation of the operations in the game (e.g. moving and rotating blocks). You should read this file and understand what representation is used for the game.
- **blocks.h/blocks.c** – blocks.c contains the definition of the blocks used in the game and operations on them.
- **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) in a queue that can then be queried.
- **ledmatrix.h/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.h/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.h/scrolling_char_display.c** – this contains code which provides a scrolling message display on the LED matrix board.
- **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** – this module encapsulates all SPI communication. Note that by default, all SPI communication uses busy waiting – 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 way 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 is available 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.

Initial Operation

The provided program responds to the following inputs:

- Rising edge on the button connected to pin B3 (moves block left – if possible)
- Rising edge on the button connected to pin B2 (rotates block clockwise – if possible)
- Rising edge on the button connected to pin B1 (moves block down one row – if possible)
- Serial input escape sequence corresponding to the left cursor key (moves block left – if possible)
- Serial input escape sequence corresponding to the up cursor key (rotates block clockwise – if possible)
- Serial input escape sequence corresponding to the down cursor key (moves block down one row – if possible)

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

- Rising edge on the button connected to pin B0 (intended to move block right);
- Serial input escape sequence corresponding to the cursor right key (also move block right).
- Serial input characters 'p' and 'P' (intended to be the pause/unpause key)

Program Features

Marks will be awarded for the features described below. (The marks available for individuals and groups are shown on the feature summary page.) 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 that 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 falling block to be moved left, rotated and dropped. 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 must 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*.

Move Block Right (Level 1)

The provided program will only move the falling block to the left. You must complete the `move_block_right()` function in file *blocks.c*. This function must not allow the block to move off the board area.

Scoring #1 (Level 1)

Add a scoring method to the program as follows:

- 1 is added to the score each time a block is fixed to the board (i.e. descends as low as it can go).
- 100 is added to the score each time a row is completed and cleared.

(Full marks for this feature depends on other level 1 features being implemented – i.e. being able to clear completed rows.) You should make use of the function `add_to_score( uint16_t 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 must also add code to display the score (to the serial terminal in a fixed position) only when it changes. The displayed score must be right-aligned – i.e. the right-most digit in the score must be in a fixed position.

Drop Block from Height (Level 1)

The provided program does not allow blocks to be dropped all the way to the bottom in one move – they only drop one row at a time – either periodically or when a particular key is pressed. Modify the code so that pressing either button B1 or the space bar on the serial terminal causes the block to be immediately dropped as far as it can. (The cursor down key should still move the block down just one row.) One approach would be to repeatedly call `attempt_drop_block_one_row()` until it fails.

Clear Completed Rows (Level 1)

Every time a row is completed (i.e. there is a solid row of blocks across the board) the row should be removed and the blocks above should drop down. Complete the function `check_for_completed_rows()` in file *game.c*. If the same block completes multiple rows then this must be handled. This is the most difficult of the Level 1 features.

Game Pause (Level 1)

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 except the 'n' or 'N' keys if "New Game" is implemented as described below.) The block dropping rate must be unaffected – if the pause happens 450ms before a block drop is due, then the drop should not happen until 450ms after the game is resumed, not immediately upon resume.) The check for this key press is implemented in the supplied code, but does nothing.

New Game (Level 1 – group feature)

Add a feature so that if the 'n' or 'N' key is pressed, a new game is started (at any time – including game over or if the current game is in progress but not if leaderboard initials are being entered for the level 3 feature described below). The board must be cleared and the score reset. Game play should behave as it does for the first game after power-on.

Additional Block Shapes (Level 1 – group feature)

Modify the program to support at least two additional block shapes. You will need to modify *blocks.c* appropriately – including the `NUM_BLOCKS_IN_LIBRARY` definition.

Random block position and rotation (Level 1 – group feature)

The provided program always initially places blocks at the right hand side of the board (at the top row). Modify the program so that it places the block in a random column (still at the top of the board) with a random rotation. You will need to add appropriate code to the function `generate_random_block()` in the file *blocks.c*. (The block must, of course, not extend beyond the edges of the board but it must be apparent that sometimes blocks appear at the left hand side of the board with any rotation.)

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.) You need only store scores in RAM (i.e. they will be reset when the microcontroller is reset) – you do not need to store scores persistently (EEPROM). The current high score must be displayed on each game-over.

Block Preview (Level 2)

Whilst the current block is dropping, add a display of the next block to be dropped to the terminal output – in the orientation that it will appear (e.g. if you have implemented random block rotation as above then the preview must reflect this rotation). You must use terminal block graphics.

Acceleration (Level 2)

Make the game speed up as the score gets higher. A suggested approach is that every time a row is completed (and cleared) the game speeds up *slightly*, i.e. the blocks drop 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 30 seconds.)

Count Cleared Rows (Level 2)

Add support for a seven-segment display that shows a count of the rows that are completed and cleared. (This functionality requires that the clearing of completed rows feature has been implemented.) The count should be a two digit number (0 to 99) with both digits displayed (although you may choose not to display the left digit for counts in the range 0 to 9 if you wish). The count should start at 0 and be incremented by 1 every time a row is cleared. The count should be capped at 99. It must be possible to play your game so that at least 10 rows are cleared (i.e. both digits in this count change). The display must be maintained when the game is paused. Your feature summary form must indicate which AVR pins the seven-segment display pins are connected to.

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. (See the `srandom()` 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.

Scoring #2 (Level 2 – group feature)

Add additional scoring features:

- If a block is dropped from height then 1 + the number of rows dropped must be added to the score.

- If multiple (n) rows are completed by the same block then 100 x n^2 must be added to score. (This is instead of 100 per completed row.)

Auto-repeat (Level 2 – group feature)

Add auto-repeat support to the push buttons (B0 to B3) – i.e., if they are held down then, after a short delay (e.g. 300ms), they should act as if they are being repeatedly pressed (e.g. every 50ms).

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. (You must handle the situation of the EEPROM initially containing data other than that written by your program. You will need to use a "signature" value to indicate whether your program has initialized the EEPROM for use.) Name/initial entry must be resilient to arbitrary responses (i.e. invalid characters/keypresses should not cause unexpected behaviour).

Sound Effects (Level 3)

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

- block moving left or right or being rotated
- block falling one row due to timer
- block being dropped to the bottom
- row completed and cleared
- game start-up
- constant background tune

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 switch is connected to and 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.

Joystick Support (Level 3)

Add support to use the joystick to move blocks in the same way that the serial terminal cursor keys work:

- joystick left – moves the block left (if possible)
- joystick right – moves the block right (if possible)
- joystick up – rotates the block (if possible)
- joystick down – drops the block one row

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 and quickly 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 from your game.

Game Display on Terminal Screen (Level 3)

Display a copy of the LED matrix display on the serial terminal using block graphics of various colours – possibly different colours to those used on the LED matrix. This should allow the game to be played either by looking at the LED matrix or at the serial terminal. (The serial terminal display must keep up with the matrix display.)

Advanced Feature(s) of Your Choice (Level 3)

Feel free to implement other advanced features. The number of marks that 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. 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). 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 20 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 or 2 marks towards level 3.

Submission Details

The due date for the project is **5pm Friday 3 June 2016**. 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 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. **A penalty of 4 marks will apply if you violate this requirement.**

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. **A penalty of 4 marks will apply if you violate this requirement.** (These penalties are cumulative – e.g. if you submit a zip file inside a rar file you will incur an 8 mark penalty. These submission requirements are in place to facilitate automatic processing of submissions. Incorrectly formatted submissions which require manual processing will be penalised)

If you make more than one submission, each submission must be complete – the single zip file must contain the feature summary form and 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 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 (from Tuesday 7 June 2016). 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 in Atmel Studio 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.** 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 Atmel Studio 7, 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 Atmel Studio) and check for warnings in the "Error List" tab.

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 4 June, 2016) 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 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.

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, 2016 – 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							Serial RX	Serial TX
							Baud rate: 19200	
Notes for Marker e.g. compile/link options								

Feature **(For Groups)**	✓ if attempted	**Comment** **(Anything you want the marker to consider or know?)**	**Marks** **(indiv/grp)**	
Splash screen			4/3	
Move block right			8/5	
Scoring #1			10/7	
Drop from height			10/7	
Clear rows			15/12	
Game Pause			8/5	
New Game			(1)/4	
Additional Shapes			(1)/4	
Random Position			(1)/4	
High Score			(1)/4	/55
Block Preview			8/5	
Acceleration			8/5	
Count clear rows			9/6	
Random seeding			(1)/3	
Scoring #2			(1)/2	
Auto-repeat			(2)/4	/25
EEPROM Leaders			7/5	
Sound Effects			7/5	
Joystick			7/5	
EEPROM game			7/5	
Terminal Display			7/5	
Other Advanced			max 7/7	/20 max

(Penalties apply for compile warnings, incorrect files, etc.) **Total:** (out of 100, max 100)