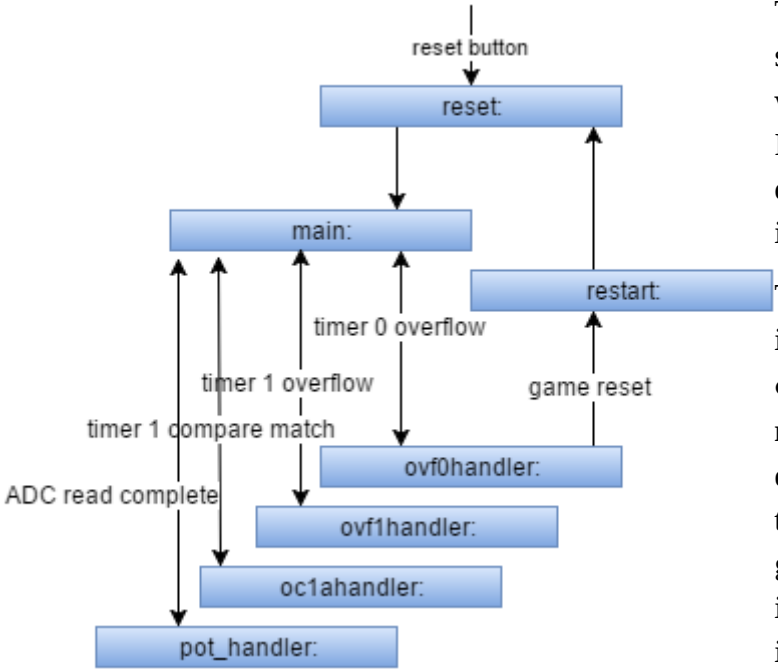
Design Manual

Table of Contents

1. System Flow Control

The safe cracker is split into a main file, `cracker.asm`, and helper files which provide functions, macros and interrupt service routines to interact with the AVR board's inputs and outputs. The helper files are described in the 'Modules' section.

The main file, `cracker.asm` implements the flow of game.



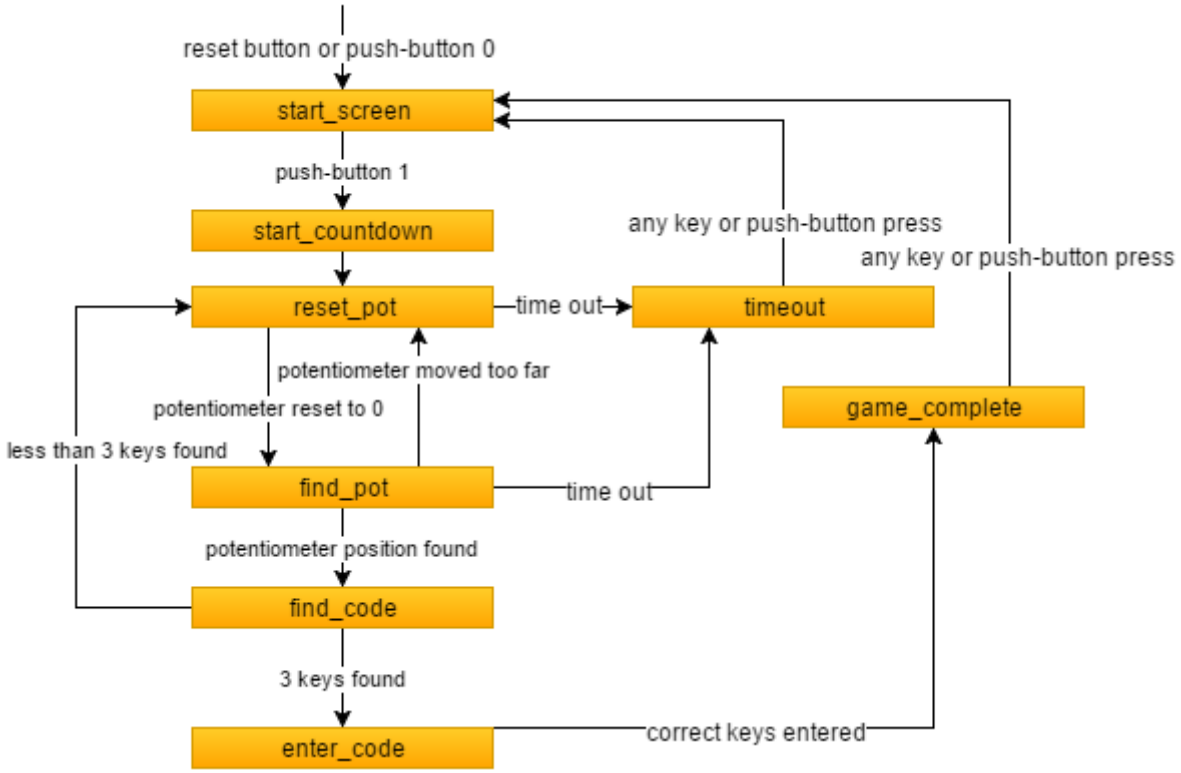
The game is initialised by the reset interrupt service routine. It then enters the main loop which does nothing but wait for interrupts. Interrupts are shown in the diagram with double ended arrows representing the interrupt and return from interrupt.

The interrupt service routine which implements the overall game flow is `ovf0handler`, the interrupt service routine for the timer 0 overflow interrupt. It executes different sections of code based on the current stage of the game. When the game is over, or if push-button 0 is pushed, it calls restart to restart the game. Otherwise it returns from interrupt to main.

`ovf0handler` implements the game flow by executing different sections of code based on the stage of the game. The game flow is simply as described by the project specifications.

`ov1handler` and `oc1ahandler` are interrupt service routines for timer 1 overflow and timer 1 compare match interrupts respectively. These are used to implement LCD fading.

`pot_handler` is the interrupt service routine for the ADC read complete interrupt, used to implement reading from the potentiometer.



2. Data Structures

Bytes are 8-bit integers. *Words* are 16-bit (or 2-byte) integers. *Characters* are represented as bytes and use ASCII character values. *Booleans* are stored as bytes.
Key presses and the difficulty level are represented as a character. Letters used in this way are capitals (eg the lowest difficulty is `'A'`).

`2.1 cracker.asm` Data Structures

Registers

`timer0_parity` is `r11`, it flips between `1` and `0` on each timer 0 overflow interrupt. It is used to control the fade speed of the LCD.

`fade_dir` is r12, it stores `-1`, `0` or `1` to control whether the LCD fades in or out.

`lcd_brightness` is `r13`, it stores the current LCD brightness from `0x00` for off to `0xff` for full brightness.

`cur_code_char` is `r14`, it stores the index of the current character being entered in the enter_code stage.

`game_iter` is `r15`, it stores how many rounds have been won/how many secret keys have been found.

`tmp` is `r16`, it is used as a miscellaneous temporary variable.

`do_display` is `r22`, it stores `1` if the display needs to be refreshed and `0` otherwise.

`stage` is `r23`, it stores which stage the stage is at, the stages are defined as constant integer symbols.

`tmp_wordl` is `r24` and `tmp_wordh` is `r25`, they are used as a miscellaneous word temporary variable.

Compiler constant symbols

`ticks_per_sec` is the approximate number of timer 0 overflows per second.

`lcd_fade_time` is the number of timer 0 overflows to wait before beginning to fade the LCD.

`pot_eps` is a small value, below which any potentiometer readings are essentially `0`.

`start_screen`, `start_countdown`, `reset_pot`, `find_pot`, `find_code`, `enter_code`, `game_complete` and `timeout` are given the values `0` through `7`.

Data segment variables

`diff_time` is a byte that stores the number of seconds the user has to find the correct potentiometer position in the current difficulty.

`diff_level` is a character that stores the current difficulty level (`'A'` to `'D'`).

`timer_cd` is word that stores the number of timer 0 overflow interrupts until a second has passed.

`timer_cnt` is a byte that stores the number of seconds left on the timer.

`pot_cd` is a word that stores that number of timer 0 overflow interrupts remaining until the potentiometer has been held in the correct position for long enough.

`pot_targ` is a word that stores the secret target value for the potentiometer.

`key_targ` is a character that stores the secret target value for the keypad.

`key_cd` is a word that stores the number of timer 0 overflow interrupts remaining until the secret key has been held for long enough.

`key_code` is 3 characters that stores the 3 letter code.

`strobe_cd` is a word that stores the number of timer 0 overflow interrupts remaining until the strobe should be toggled.

`lcd_off_cd` is a word that stores the number of timer 0 overflow interrupts remaining until the lcd should begin fading.

`random_timer` is a byte that increments on each timer 0 overflow interrupt and is used to seed the random number generator.

Program memory constants

The fixed display strings, as specified in the project specifications, are stored as program memory constants. In addition, strings to display each difficulty level, `“ (?)”` (with `?` replaced by the difficulty level as a character) are also stored as program memory constants.

2.2 Module Data Structures

The interface to the following helper files is described under 'Modules'. The algorithms used are described in 'Algorithms'. In this section, the data structures used are described.

keypad-util.asm

`pressed_char` is a character storing the last pressed character for debouncing purposes.

`key_db` is a byte storing the remaining debounce duration.

lcd-fader.asm

LCD fading implicitly stores the LCD brightness in the compare interrupt value for timer 1.

pot-util.asm

`pot` is a word that stores the potentiometer reading.

`potav` is a boolean which stores whether a new potentiometer reading is available.

rand.asm

`rand_cur` is a byte that stores the current random value for use in the linear congruential generator.

speaker-util.asm

`speaker_what` is a byte that stores `0` or `1` for whether the pin is currently high or low.

`speaker_len` is a word that stores the remaining speaking duration.

3. Algorithms

`3.1 cracker.asm` Algorithms

`reset` initialises data structure values and initialises the stack and the inputs and outputs for the board (generally by calling the initialisation functions defined in the helper file modules).

`main` sleeps and waits for interrupts.

`restart` provides a method for manual resetting through software. It disables interrupts and calls `reset`.

`ovf0handler` handles the overall game flow. While it has many cases, the code is just simple branching to execute different sections of code for each game stage. A brief description is provided below.

Code Executed on Every Timer 0 Overflow Interrupt

`timer0_parity` is flipped.

`speaker_speak` is called to make sound if there is a current beep that is not complete.

`random_timer` is incremented.

If push-button 0 is pressed, the game is restarted.

LCD Fading

If the current game stage is one where the LCD should fade and no key has been pressed for `lcd_fade_time` ticks, the LCD is set to fade. Otherwise it is set to be fully on. The LCD brightness is only changed if `timer0_parity` is `1`, meaning it will fade over approximately 500ms.

Start Screen

If the current stage is the start screen stage:

- If a letter key is pressed, the difficulty is changed.
- If push-button 1 is pressed, the stage is advanced to the start countdown stage, the speaker is set to beep for 250ms, the random number generator is seeded with the current value of `random_timer`, and `timer_cnt` is set to `3`.

Start Countdown

If the current stage is the start countdown stage:

- Any new message is displayed.
- If the time left is decremented and is not `0`, the speaker is set to beep for 250ms.
- If the time left is `0`, the stage is advanced to the reset potentiometer stage, `timer_cnt` (the time the player has remaining) is set to the current difficulty time, the speaker is set to beep for 500ms, and `pot_cd` (the duration which the pot needs to be held at `0` for) is set to 500ms of ticks.

Reset Potentiometer and Find Potentiometer timing out

If the current stage is the reset potentiometer stage or the find potentiometer stage and the time left is `0`, the game advances to the time out (game over) stage.

Reset Potentiometer

If the current stage is the reset potentiometer stage and the time left is not `0`:

- Any new message is displayed.
- If the potentiometer reading is not `0`, `pot_cd` is set to 500ms of ticks.
- Otherwise `pot_cd` is decremented.
- If `pot_cd` is `0`, the stage is advanced to the find potentiometer stage, `pot_cd` (now the duration for which the potentiometer must be held in the correct position) is set to 1 second of ticks and the target potentiometer position is set to a random value.

Find Potentiometer

If the current stage is the find potentiometer stage and the time left is not `0`:

- Any new message is displayed.
- If a potentiometer reading is available
    - if the reading is past the correct position, the stage is set back to the reset potentiometer stage and `pot_cd` is reset to 500ms of ticks
        - Otherwise, the LEDs are set to the correct configuration.
        - If the potentiometer is not at the target, `pot_cd` is reset to 1 second of ticks.
        - Otherwise, `pot_cd` is decremented.
        - If `pot_cd` is `0`, the stage is advanced to the find code stage, the LEDs are cleared, the target key is set to a random character and `key_cd` (the duration which the correct key needs to be held for) is set to 1 second of ticks.

Find Code

If the current stage is the find code stage:

- Any new message is displayed.
- If the correct key is not held, `key_cd` is reset to 1 second of ticks.
- Otherwise `key_cd` is decremented.
- If `key_cd` is `0`, the stage advances depending on `game_iter`. If `game_iter` is `3`, the stage is advanced to the enter code stage and `cur_code_char` (the number of correct keys entered) is reset to `0`. Otherwise, the stage is set back to the reset potentiometer stage, `timer_cnt` is set to the current difficulty time and `pot_cd` is set to 500ms of ticks.

Enter Code

If the current stage is the enter code stage:

- Any new message is displayed.
- If an incorrect key is entered, `cur_code_char` is reset to `0`.
- Otherwise, the current number of correct key is displayed as a number of asterisks. If all three correct keys are entered, the stage is advanced to the game complete stage.

Time Out and Game Complete Game Restart

If the current stage is the time out or game complete stage and any key or push-button is pressed, the game is restarted.

Game Complete

If the current stage is the game complete stage:

- The game complete message is displayed (once) and the speaker is set to beep for 1 second (once).
- The strobe is toggled based on `strobe_cd`.

Time Out

If the current stage is the time out stage:

- The game over message is displayed (once).

3.2 Module Algorithms

The interface to the following helper files is described under 'Modules'. The data structures used are described in 'Data Structures'. In this section, the non-trivial algorithms used in the helper files are described.

keypad-util.asm

`read_key` is implemented simply by looping over each row of the keypad and then over the columns of that row.

`read_key_db` implements debouncing by storing the last key pressed, and keeping a counter of how many times a different key or no key needs to be pressed to register new key presses. Once the counter reaches 0, new key presses will be registered.

lcd-fader.asm

LCD brightness is implemented by turning the LCD on and off, but at small intervals so that the LCD appears to just be dimmer. The greater the proportion of time the LCD is on, the brighter it appears.

Specifically, `set_lcd_level` sets the proportion of each timer 1 overflow that the LCD will be on. If the level is 5 or lower, the brightness is effectively 0 so the LCD is turned off. Otherwise, each time timer 1 overflows, the LCD is turned on. And each time timer 1 reaches the set level, the LCD is turned off. (Thus a higher LCD brightness level will cause the LCD to be on for a larger proportion of each overflow.)

lcd-util.asm

`print_int` is implemented by printing each digit in order: the hundreds digit, then the tens digit then the ones digit, taking care to ignore leading 0s but not non-leading 0s.

pot-util.asm

`pot_handler` is the interrupt service routine for the ADC read complete interrupt. It stores potentiometer readings and whether a new reading is available. Then, `pot_read` simply returns the value of the new reading if one is available (and an invalid value otherwise) and then requests a new reading if needed.

rand.asm

`rand` is implemented using a simple linear congruential generator. To create different starting values, the generator is seeded with a timer value when the user begins a new game. Since the timer overflows very often, the seed value is generally random enough.

speaker-util.asm

Speakers are implemented by simply storing the remaining beep duration. `speaker_set_len` sets the remaining duration. Then, `speaker_speak` is run on each timer0 overflow interrupt and switches the pin only if the current beep is not complete.

4. Module Specifications

In addition to the main file, a number of helper files define functions, macros and interrupt service routines to interact with the AVR board's inputs and outputs. Only functions, macros and interrupt service routines used in `cracker.asm` are described.

Macro arguments are referred to as `@x` (as in avr assembly). `a:b` represents a pair of registers representing a word, where `a` is the more significant byte.

keypad-util.asm

`keypad_init` (macro): Initialises the keypad.

`read_key` (function): Returns a key which is currently pressed as a character in `r16`. If no key is pressed, it returns `'?'`.

`read_key_db` (function): Returns a key which is currently pressed as a character in `r16` with debouncing. If no key is pressed (or not pressed enough to pass debouncing) it returns `'?'`.

lcd-fader.asm

`lcd_fade_init` (macro): Sets up the LCD for fading.

`set_lcd_level` (function): Sets the brightness level of the lcd based on `r16`. When `r16` is `0x00` brightness is lowest, when `r16` is `0xff` brightness is highest.

`ovf1handler` (interrupt service routine): Interrupt service routine for timer 1 overflow interrupt for LCD fading.

`oc1ahandler` (interrupt service routine): Interrupt service routine for timer 1 compare match interrupt for LCD fading.

lcd-util.asm

`do_lcd_data` (macro): Prints immediate character `@0` to the LCD.

`lcd_row2` (macro): Moves printing on the LCD to the second row.

`lcd_clear` (macro): Clears the LCD (and moves printing to the first row).

`lcd_init` (macro): Initialises the LCD.

`rip` (function): Prints `'*'` and infinite loops. Should never be called.

`print_int` (function): Prints `r16` as an integer to the LCD.

led-util.asm

`set_led` (macro): Displays the lowest 10 bits of immediate `@0` on the LEDs. An LED is turned on if the corresponding bit is set (if the bit is `1`). The most significant bit is displayed on the top LED.

motor-util.asm

`motor_init` (macro): Initialises the motor.

`set_motor_speed` (macro): Turns the motor off if immediate `@0` is `0`, otherwise turns the motor to full speed.

pb-util.asm

`ispb0` (macro): Sets the Z flag to `1` if push-button 0 is pressed. Otherwise sets it to `0`.

`ispb1` (macro): Sets the Z flag to `1` if push-button 1 is pressed. Otherwise sets it to `0`.

`is_any_keys` (function): Sets the Z flag to `1` if push-button 1 is pressed or any key on the keypad is pressed. Otherwise sets it to `0`.

pot-util.asm

`pot_req` (function): Initialises the potentiometer. (Used internally to request a reading from the potentiometer.)

`pot_read` (function): Returns the potentiometer reading in `r17:r16`. If no reading is available, it returns `0xffff`.

`pot_handler` (interrupt service routine): Interrupt service routine for ADC read complete interrupt for reading the potentiometer.

print-string.asm

`puts` (macro): Prints the string at immediate program memory *word* address `@0` to the LCD.

rand.asm

`srand` (macro): Seeds the random number generator with `r16`.

`rand` (function): Returns a pseudo-random 8-bit integer in `r16`.

`rand_char` (function): Returns a (pseudo-)random character on the keypad.

speaker-util.asm

`speaker_init` (macro): Initialises the speaker.

`speaker_speak` (macro): Switches the speaker pin between high and low while the current beep is not completed.

`speaker_set_len` (macro): Sets the remaining beep duration.

util.asm

`read_word` (macro): reads a word from immediate data memory address `@2` to registers `@0:@1`.

`write_word` (macro): writes a word from registers `@0:@1` to immediate data memory address `@2`.

`write_const_word` (macro): writes immediate word `@1` to immediate data memory address `@2`.

`subi_word` (macro): subtracts immediate byte `@2` from registers `@0:@1`. Neither register can be `r16`.

`dec_word` (macro): decrement from registers `@0:@1`. Neither register can be `r16`.

`def_string` (macro): places immediate string `@0` in program memory with correct padding.