Design Manual

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.
[flowchart]

Data Structures

Bytes are 8-bit integers. *Words* are 16-bit (or 2-byte) integers. *Characters* are (represented as, and can be considered as) bytes. *Booleans* are stored as bytes.
Key presses and the difficulty level are represented as a character. Letters in this case are represented as capitals (eg the lowest difficulty is 'A').

cracker.asm

blah

blah

blah

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.

Algorithms

cracker.asm

blah

blah

blah

The interface to the following helper files is described under 'Modules'. 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 made the handler of the ADC read complete interrupt. It stores read potentiometer values and stores whether that a new read value is available. Then, pot_read simply returns a new read value 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 duration to speak, and making sound while the duration is not up. speaker_set_len sets the remaining duration. Then, speaker_speak is run every time timer0 overflows and switches the pin only if there is duration remaining.

Modules

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): Handler for timer 1 overflow interrupt for LCD fading.

oc1ahandler (interrupt service routine): Handler 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): Handler 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 if there is speaking duration remaining.

speaker_set_len (macro): Sets the remaining duration to make sound.

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.

…

…

…

blahblah