Microwave Oven Simulator - Conceptual Design and Development

Author: Andrew Wong (z5206677)

Introduction

This project aims to create a simulated Microwave Oven with an ATMega2560 AVR board.

A successful implementation of a simulated microwave oven requires the simulation of a turntable, timer, magnetron, operational buttons, a door, and a display.

State Table

The below table outlines the functionality, and operations that the microwave oven will perform depending on its current state.

	ENTRY	RUNNING	PAUSED	FINISHED	PWR_CFG	DOOR_OPEN
Turntable	-	ROTATE	-	-	-	-
Countdown	-	TIMER	-	-	-	-
Magnetron	-	ACTIVE	-	-	-	-
Intensity LED	-	ON	-	-	-	-
Time	SHOW	SHOW	SHOW	-	SHOW	SHOW
Door	C	C	C	C	C	O
LCD	-	-	-	DONE	PWR	-
Brightness	FADE	ON	FADE	FADE	FADE	-
Door LED	-	-	-	-	-	ON
[0]-[9]	INPUT	-	-	-	1/2/3	-
[START]	START	ADD	RESUME	-	-	-
[STOP]	RESET	PAUSE	RESET	RESET	BACK	-
[A]	PWR	-	-	-	-	-
[OPEN]	DOOR	PAUSE -> DOOR	DOOR	RESET -> DOOR	DOOR	-
[CLOSE]	-	-	-	-	-	CLOSE

State Operation

- The `ENTRY` state waits for the user to enter a time with the numerical keypad (0-9).

    - Pressing [START] will enter the `RUNNING` state
    - There is a maximum of 99 minutes and 59 seconds of continual running operation
    - If no time was entered, the microwave will automatically start a timer for 1 minute (60 seconds)
    - Pressing [A] will enter the `SET PWR` menu mode
    - Pressing [1] will set the magnetron intensity to 100%
    - Pressing [2] will set the magnetron intensity to 75%
    - Pressing [3] will set the magnetron intensity to 50%
    - After [1]/[2]/[3]/[#] is pressed, the microwave will return to the `ENTRY` menu
    - Pressing [STOP] will reset the timer to 0 seconds
- The `RUNNING` state simulates the cooking of the food
    - The turntable is rotated
    - The magnetron is activated
    - The timer is activated
    - Pressing [START] will add another minute (60 seconds) to the timer
- The `PAUSED` state simulates a temporary halt in cooking operation
    - The turntable is stopped
    - The magnetron is deactivated
    - The timer is deactivated
    - Pressing [START] will cause the microwave to resume cooking in the `RUNNING` state
    - Pressing [STOP] will cancel the cooking job, and the microwave will return to the `ENTRY` state
- The `FINISHED` state simulates a finished cooking operation
    - This state occurs when the timer reaches 0
    - The turntable is stopped
    - The magnetron is deactivated
    - The timer is deactivated
    - Pressing [STOP] or [OPEN] will return the microwave to the `ENTRY` state
- When the magnetron is activated, the LED bars will show the intensity of the magnetron
- At all times, pressing [OPEN] will halt all operations
    - The door LED will activate
    - No button input will work until the [CLOSE] button is pressed
    - The door LED will deactivate
    - The microwave will then resume operation
    - If the microwave was in the `RUNNING` state, the state will first change to `PAUSED`
    - If the microwave was in the `FINISHED` state, the state will first change to `ENTRY`
- At all times, if the backlight is dimmed then any button press will wake up the display
- The timer causes a counter to decrement every second
    - Every time the counter decrements, the new time is displayed on the LCD screen (top-left)
    - If the timer reaches zero, the microwave enters the `FINISHED` state

Implementation

Design Decisions / Rationale

Hardware Usage

- Timer 0 - Debouncing
- Timer 1 - Central clock
- Timer 3 - PWM controller motor
- Timer 5 - PWM controlled LCD backlight
- Port A, F - LCD Screen
- Port C - LED Bar
- Port D - Push Buttons
- Port E - Motor
  - Used for OC3B pin
- Port G - LED
  - *Door LED*
- Port K - Keypad
  - Used because they are also Pin Change Interrupt pins
  - Can be used to trigger the PCINT2 interrupt
- Port L - LCD Screen Light
  - Used for OC5B pin

Door Open LED

When the microwave door is open, the STROBE LED will be turned on continually, rather than the top-most LED of the LED bar.
`LED9` can be mapped to `PG2` instead, if the top-most LED of the LED bar is desired

Input Design

I have decided to implement all buttons to trigger as an interrupt - as like an event-based approach.
To mitigate switch bouncing - **All buttons will share a single software debouncer** (`timer0`).
Whilst the OPEN and CLOSE buttons don't *really* need to be debounced, we'll do it anyway.

Time Input

The third digit of the timer (tens column) is limited from 0-5.
As a result, the maximum possible time of cooking is 99:59 (99 minutes, 59 seconds).

The add minute operation (Pressing [Start] while running) will be dismissed if there is 99 minutes remaining.

Interrupts

To mitigate interrupt events modifying registers from other operations, interrupts share their own general purpose register `r18` ; rather than using `r17` .

Consequently, there is a `isRunning` and `isRunningISR` macro.

Keypad

The example keypad checking code (Lab 4) executes under the guise that it will be run continually in a loop.

This implementation of the microwave will migrate this code into an interrupt-based routine.

The keypads buttons have been assigned to `Port K` instead - which also serves as `PCINT23:16` pins.
Using `Pin Change Interrupt 2`, keypad presses will trigger the `PCINT2` interrupt

Backlight

When the backlight is dimmed, any keypad button will reactivate the display - however the keypad function will be dismissed if invalid at the time of press.
The Open and Close buttons will not (by design) activate the backlight

Timer

`timer0` serves as a debouncing timer

`timer1` serves as a multipurpose central clock operating at 12Hz.
This timer supports turntable rotation (4 states at 3 revolutions a second = 4 * 3 = 12 changes a second)
It also supports the cooking timer, which waits for 12 ticks ($^1/_{12}$ * 12 = 1 second)
Finally, the timer implements the fading LCD backlight every 6 ticks, at 8-bit duty cycle value increments of floor($^{255}/_6$) = 42.

Motor Speed

The PWM duty cycle for a motor rotation speed of 70 rps can be determined by trial and error with help of the optometer.
However, I did not have the project board in possession, and so was unable to calculate the correct scaled PWM value

Register Access

As registers are faster than the SRAM in the MCU, I have opted to store most flags, values and states in registers.
To minimise the number of registers used / mitigate conflict registers, some arithmetic operations have been used - namely for `r0` and `r21:r20`.

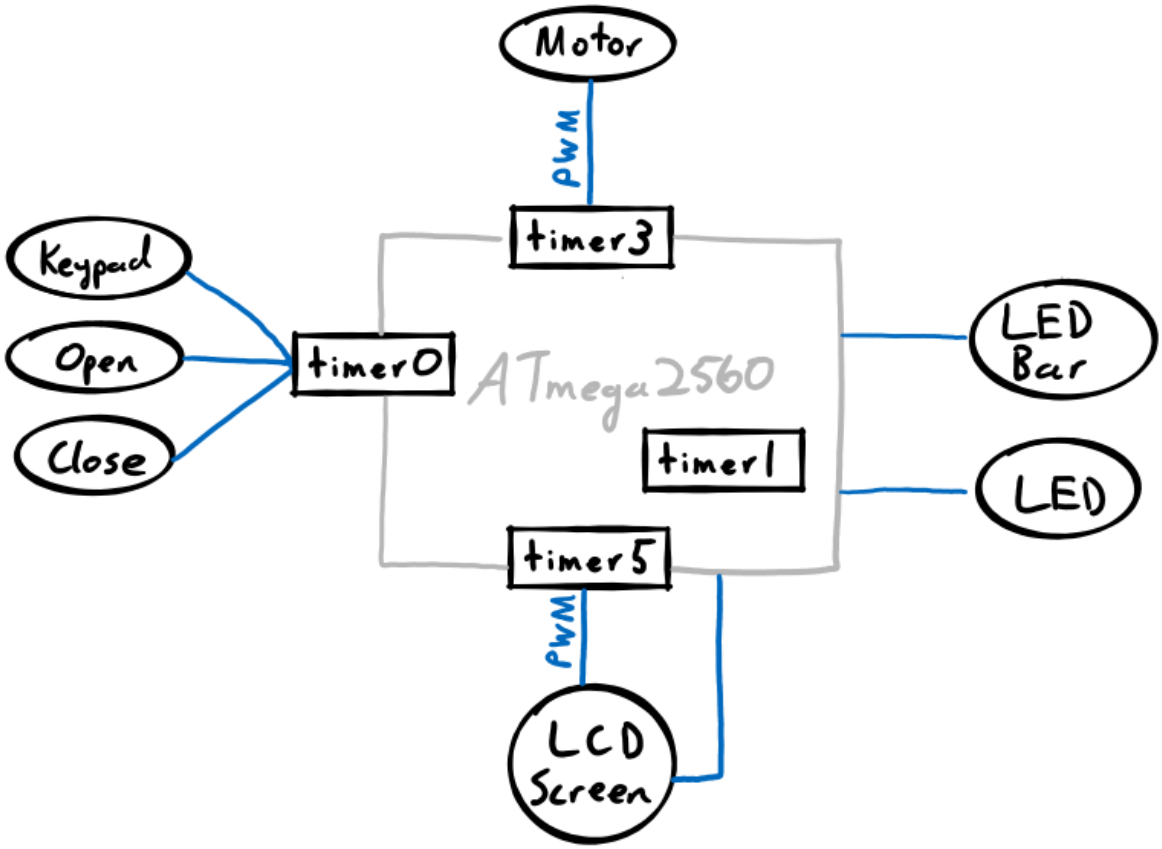
Consequently, the stack pointer (SPL) did not have to be initialised

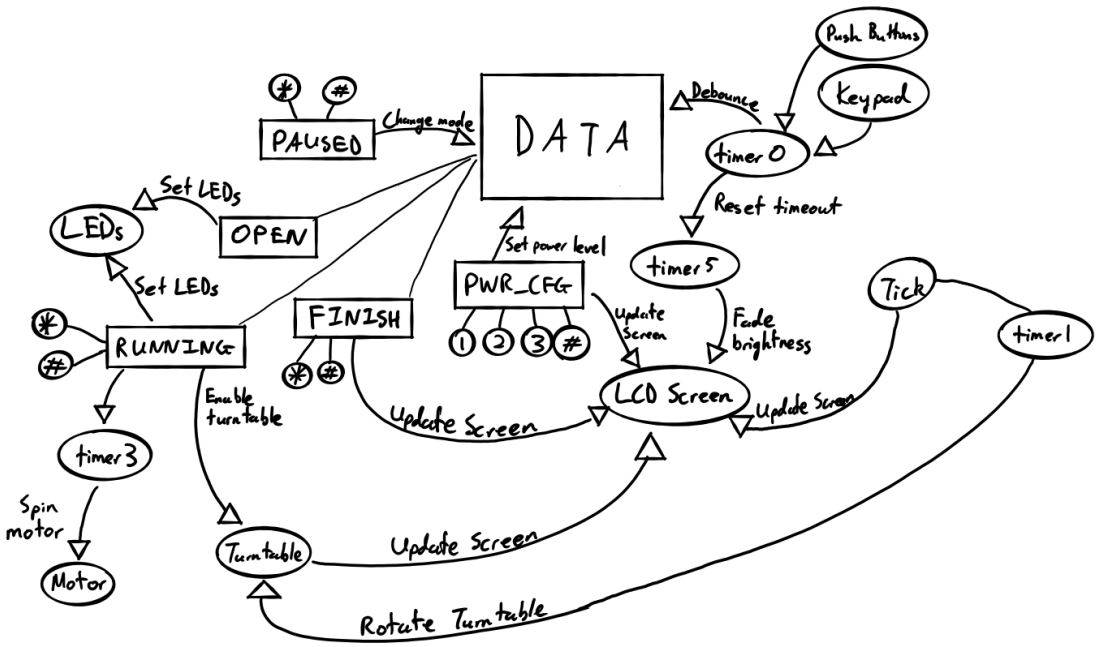
Functions and Macros

Operations have been assigned into macros and functions.
This allows the reuse of blocks of instructions, and has been used extensively.

User-triggered operations have been created as functions, whilst utility operations have been created as macros.

System Diagrams





Component-Hardware Map

Microwave Component	Hardware Device	Component Port	Board/AVR Port

Microwave Component	Hardware Device	Component Port	Board/AVR Port
Turntable	LCD	[D0-D7] [BE;RW;E;RS] [BL]	[PF0-PF7] [PA4-PA7] [PL4/OC5B]
Countdown	LCD	[D0-D7] [BE;RW;E;RS]	[PF0-PF7] [PA4-PA7]
Magnetron	MOTOR	MOT	[PE4/OC3B]
Power Level	LED	[LED0-LED7]	[PC0-PC7]
Start Button	*	KEYPAD	[PK0-PK7]
Stop Button	#	KEYPAD	[PK0-PK7]
Open Button	PUSH BUTTON	PB1	RDX3 (PD1)
Close Button	PUSH BUTTON	PB0	RDX4 (PD0)
Power Select	A	KEYPAD	[PK0-PK7]
Numeric Pad	KEYPAD	KEYPAD	[PK0-PK7]
Door Light	STROBE	LED	PG2
Door Status	LCD	[D0-D7] [BE;RW;E;RS]	[PF0-PF7] [PA4-PA7]
Status Message	LCD	[D0-D7] [BE;RW;E;RS]	[PF0-PF7] [PA4-PA7]

Button Setup

Register Usage

Category	Purpose	Register
System	System State	`r0`
Input	Input Ready Flag	`r1`
Magnetron	Level Display	`r2`
Magnetron	PWM Duty Cycle	`r3`
Input	Debouncer Temporary	`r16`
System	General Operation Temporary	`r17`
System	General Routine Temporary	`r18`
Input	Duration Temporary	`r19`
Time	Duration	`r21:r20`
Time	Timer Temporary	`r22`
Time	Seconds Counter Temporary	`r23`

Category	Purpose	Register
Turntable	Turntable Position	`r24`
LCD	Backlight Timeout Counter	`r25`
LCD	Backlight PWM Duty Cycle	`r26`

System State - `r0`

7	6	5	4	3	2	1	0
-	-	DOOR_OPEN	POWER_CFG	FINISHED	PAUSED	RUNNING	ENTRY

- Bits 7-6 - undefined
- Bit 5 defines if the door is currently open
- Bits 0-4 define the current mode

Input Ready Flag - `r1`

- Zero value - Not ready for input
- Non-zero value - Ready for input

Level Display - `r2`

Each bit represents one of the lower LEDs on the LED bar connected to PORTC.

PWM Duty Cycle - `r3`

The duty cycle value to assign to the OCR3BL register during magnetron operation

Debouncer Temporary - `r16`

`timer0` -incremented counter for debouncing

General Operation Temporary - `r17`

General purpose register for operations and macro/function outputs

General Routine Temporary - `r18`

General purpose register for interrupt service routine operations

Duration Temporary - `r19`

Counts the number of digits entered in the system.
Prevents more than 4 digits from being entered

Duration - `r21:r20`

16-bit value that contains the entered/remaining time.
The 16-bit value is split into four 4-bit values (the highest digit `9` needs at most 4 bits).

r21:7-4	r21:3-0	r20:7-4	r20:3-0
m	m	s	s

> i.e. `0b1001000110100`
> `0001` (1) … `0010` (2) … `0011` (3) … `0100` (4)
> = 12m 34s

Timer Temporary - `r22`

`timer1` -incremented counter for counting $^1/_{12}$ seconds.

Provides functionality turntable rotation, seconds counting and backlight fading

Seconds Counter Temporary - `r23`

`timer1` -incremented counter for counting seconds.

Turntable Position - `r24`

Holds the current rotational state of the turntable

Backlight Timeout Counter - `r25`

`timer1` -incremented counter for counting to 10 seconds.

Backlight PWM Duty Cycle - `r26`

The duty cycle value to assign to the `OCR5BL` register during backlight fading

Software Implementation

Define data regions

- `turntablePositions => "-/|\"`
- `powerCfgMessage1 => "Set Power 1/2/3\0"`
- `powerCfgMessage1 => "# - Return\0"`
- `finishedMessage1 => "Done\0"`
- `finishedMessage2 => "Remove food\0"`

Interrupt Vector Table

Address	(r)jmp

Address	(r)jmp
RESET (`0x0000`)	`init`
`INT0addr`	`btnClose`
`INT1addr`	`btnOpen`
`PCINT2addr`	`btnKeypad`
`OVF1addr`	`Timer1OVF`
`OVF0addr`	`Timer0OVF`

Macro Definitions

- macro: `StartDebouncer` - *Enable button debouncer*
  - Set the `r1` register to `0` - Disable button input
  - Set the `TCNT0` register to `0` - Clear the timer
  - Set the `r16` register to `0` - Clear the previous ticks
  - Set the `TIMSK0` register to `0b1` - Enable timer
- macro: `StopDebouncer` - *Disable button debouncer*
  - Set the `TIMSK0` register to `0b0` - Disable timer
  - Set the `r1` register to `1` - Enable button input
- macro: `DoorLEDOn` - *Enable Door LED*
  - Get the value of the `PORTG` register
  - Bitwise OR the value with `0b100` - Enable G2
  - Store the new value into the `PORTG` register
- macro: `DoorLEDOff` - *Disable Door LED*
  - Get the value of the `PORTG` register
  - Bitwise AND the value with `0b11111011` - Disable G2
  - Store the new value into the `PORTG` register
- macro: `isEntry` - *Check if ENTRY mode*
  - Copy `r0` to `r17`
  - Bitwise AND `r17` with `0b1`
- macro: `isRunning` - *Check if RUNNING mode*
  - Copy `r0` to `r17`
  - Bitwise AND `r17` with `0b10`
- macro: `isRunningISR` *(Use register r18 for ISR)*
  - Copy `r0` to `r18`
  - Bitwise AND `r18` with `0b10`
- macro: `isPaused` - *Check if PAUSED mode*

    - Copy `r0` to `r17`
    - Bitwise AND `r17` with `0b100`

- macro: `isFinished` - *Check if FINISHED mode*

    - Copy `r0` to `r17`
    - Bitwise AND `r17` with `0b1000`

- macro: `isPowerCfg` - *Check if PWR_CFG mode*

    - Copy `r0` to `r17`
    - Bitwise AND `r17` with `0b10000`

- macro: `setEntry` - *Set mode to ENTRY*

    - Set `r17` to `r0`
    - Bitwise AND `r17` with `0b11100000`
    - Bitwise OR `r17` with `0b1`
    - Set `r0` to `r17`

- macro: `setRunning` - *Set mode to RUNNING*

    - Set `r17` to `r0`
    - Bitwise AND `r17` with `0b11100000`
    - Bitwise OR `r17` with `0b10`
    - Set `r0` to `r17`

- macro: `setPaused` - *Set mode to PAUSED*

    - Set `r17` to `r0`
    - Bitwise AND `r17` with `0b11100000`
    - Bitwise OR `r17` with `0b100`
    - Set `r0` to `r17`

- macro: `setFinished` - *Set mode to FINISHED*

    - Set `r17` to `r0`
    - Bitwise AND `r17` with `0b11100000`
    - Bitwise OR `r17` with `0b1000`
    - Set `r0` to `r17`

- macro: `setPowerCfg` - *Set mode to PWR_CFG*

    - Set `r17` to `r0`
    - Bitwise AND `r17` with `0b11100000`
    - Bitwise OR `r17` with `0b10000`
    - Set `r0` to `r17`

- macro: `isDoorOpen` - *Check if door is open*

    - Copy `r0` to `r17`
    - Bitwise AND `r17` with `0b100000`

- macro: `subtract1` - *Subtract a second with carry*

    - Set `r18` to `r20`
    - Bitwise AND `r18` with `0b0001111`
    - If `r18` is `0`
    - Bitwise AND `r20` with `0b11110000`
    - Stop if `r21` is `0` and `r20` is `0`
    - Bitwise OR `r20` with `9`
    - macro: `subtract10`
    - Else
    - Decrement `r20`

- macro: `subtract10` - *Subtract 10 seconds with carry*

    - Set `r18` to `r20`
    - Bitwise AND `r18` with `0b11110000`
    - If `r18` is `0`
    - Stop if `r21` is `0`
    - Bitwise AND `r20` with `0b00001111`
    - Bitwise OR `r20` with `0b01010000`
    - macro: `subtract100`
    - Else
    - Decrement `r20` by `0b00010000`

- macro: `subtract100` - *Subtract 1 minute with carry*

    - Set `r18` to `r21`
    - Bitwise AND `r18` with `0b0001111`
    - If `r18` is `0`
    - Bitwise AND `r21` with `0b11110000`
    - Stop if `r21` is `0`
    - Bitwise OR `r21` with `9`
    - macro: `subtract1000`
    - Else
    - Decrement `r21`

- macro: `subtract1000` - *Subtract 10 minutes*

    - Decrement `r21` by `0b00010000`

- macro: `addMinute` - *Add one minute*

    - Set `r17` to `r21`
    - Bitwise AND `r17` with `0b00001111`
    - If `r17` is `9`
    - Set `r17` to `r21`
    - Right shift `r17` four times
    - Stop if `r17` is `9`
    - Increment `r17`
    - Left shift `r17` four times
    - Set `r21` to `r17`
    - Else
    - Increment `r21`

- macro: `resetTime` - *Clear the time*
    - Set `r21:r20` to `0`
    - Set `r19` to `0`
    - Execute function: `showTime`

ISR:[`btnClose`]

- Disable global interrupts
- Dismiss if input not ready (`r1 != 1`)
- macro: `StartDebouncer`
- Return from interrupt (+ enable global interrupts) if macro: `isDoorOpen` is false
- Bitwise XOR `r1` with `0b100000` - Clear door open bit
- Execute macro: `DoorLEDOff`
- Set LCD Line 4 Column 16 to `'C'`
- Return from interrupt (+ enable global interrupts)

ISR:[`btnOpen`]

- Disable global interrupts
- Dismiss if input not ready (`r1 != 1`)
- macro: `StartDebouncer`
- Return from interrupt (+ enable global interrupts) if macro: `isDoorOpen` is true
- Call function: `doPause` if macro: `isRunning` is true
- Call function: `doEntry` if macro: `isFinished` is true
- Bitwise XOR `r1` with `0b100000` - Set door open bit
- Execute macro: `DoorLEDOn`
- Set LCD Line 4 Column 16 to `'O'`
- Return from interrupt (+ enable global interrupts)

ISR:[`btnKeypad`]

- Disable global interrupts
- Dismiss if input not ready (`r1 != 1`)
- Execute macro: `StartDebouncer`
- Set `r25` to `0` - Enable LCD backlight
- Return from interrupt (+ enable global interrupts) if macro: `isDoorOpen` is true
- Detect the pressed key into `r17` (*Lab 4 Keypad code*)
    - For each column
    - Set column bit on `PORTK` register to `0` (LOW - GND)
    - Read from `PINL` register into `r17`
    - Check if any row bit is `0` , and stop checks if so
    - Set column bit on `PORTK` register to `1` (HI)
    - Translate co-ordinate to ASCII character into `r17`
- If macro: `isEntry` is true
    - Call function: `doPwrCfg` if key is `0xA`
    - Execute macro: `resetTime` if key is `0xF`
    - Call function: `doRun` if key is `0xE`
    - Call function: `addTime` if key is 0 - 9

- ElseIf macro: `isRunning` is true
  - Execute macro: `addMinute` if key is `0xE`
  - Call function: `doPause` if key is `0xF`
- ElseIf macro: `isPaused` is true
  - Call function: `startRunning` if key is `0xE`
  - Execute macro: `setEntry` if key is `0xF`
- ElseIf macro: `isFinished` is true
  - Call function: `doEntry` if key is `0xF`
- ElseIf macro: `isPowerCfg` is true
  - Call function: `setPwr` if key is 1 - 3
  - Call function: `returnEntry` if key is `0xF`
- Return from interrupt (+ enable global interrupts)

function:[`doEntry`]

- Execute macro: `resetTime`
- Execute function: `returnEntry`
- Return

function:[`doPwrCfg`]

- Clear LCD line 2
- Clear LCD line 3
- Display `powerCfgMessage1` on LCD line 2
- Display `powerCfgMessage2` on LCD line 3
- Return

function:[`setPwr`]

- If `r17` is `1`
  - Set `r2` to `0b11111111`
  - Set `r3` to `0xFF`
- If `r17` is `2`
  - Set `r3` to `0b1111`
  - Set `r3` to `0x7F`
- If `r17` is `3`
  - Set `r2` to `0b11`
  - Set `r3` to `0x3F`
- Return

function:[`returnEntry`]

- Clear LCD line 2
- Clear LCD line 3
- Execute macro: `setEntry`
- Return

function:[`doRun`]

- Execute macro: `addMinute` if `r21:r20` is `0`
- Call function: `startRunning`
- Return

function:[`startRunning`]

- Execute macro: `setRunning`
- Set `r25` to `0` - Enable LCD backlight
- Set `PORTC` to `r2` - Enable Magnetron level
- Set `OCR3BL` to `r3` - Enable Magnetron
- Return

function:[`stopRunning`]

- Set `PORTC` to `r2` - Disable Magnetron level
- Set `OCR3BL` to `r3` - Disable Magnetron
- Return

function:[`doPause`]

- Execute function: `stopRunning`
- Execute macro: `setPaused`
- Return

function:[`doFinish`]

- Execute macro: `setFinished`
- Clear LCD line 2
- Clear LCD line 3
- Display `finishedMessage1` on LCD line 2
- Display `finishedMessage2` on LCD line 3
- Return

function:[`showTime`]

- If `r21:r20` is `0`
  - Set LCD Row 1 Column 1 to `' '`
  - Set LCD Row 1 Column 2 to `' '`
  - Set LCD Row 1 Column 3 to `' '`
  - Set LCD Row 1 Column 4 to `' '`
  - Set LCD Row 1 Column 5 to `' '`
  - Return
- Set `r17` to `r21`
- Right shift `r17` four times
- Increment `r17` by ASCII '0' (30)
- Set LCD Row 1 Column 1 to `r17`
- Set `r17` to `r21`
- Bitwise AND `r17` with `0b1111`
- Increment `r17` by ASCII '0' (30)

- Set LCD Row 1 Column 2 to `r17`
- Set LCD Row 1 Column 3 to `':'`
- Set `r17` to `r20`
- Right shift `r17` four times
- Increment `r17` by ASCII '0' (30)
- Set LCD Row 1 Column 4 to `r17`
- Set `r17` to `r20`
- Bitwise AND `r17` with `0b1111`
- Increment `r17` by ASCII '0' (30)
- Set LCD Row 1 Column 5 to `r17`
- Return

function:[`addTime`]

- Return if `r19` is `4`
- Return if `r17` is more than `9`
- If `r19` is 0
  - Left shift `r17` by 4
  - Set `r21` to `r17`
- ElseIf `r19` is 1
  - Bitwise AND `r21` with `0b11110000`
  - Bitwise OR `r21` with `r17`
- ElseIf `r19` is 2
  - Return if `r17` is more than 5
  - Left shift `r17` by 4
  - Set `r20` to `r17`
- ElseIf `r19` is 3
  - Bitwise AND `r20` with `0b11110000`
  - Bitwise OR `r20` with `r17`
- Increment `r19`
- Execute function: `showTime`
- Return

function:[`tickTime`]

- Return if `r21:r20` is 0
- Execute macro: `subtract1`
- Execute function: `showTime`
- Return

function:[`checkBacklight`]

Complete fade in/out will execute over 6 calls (6 / 12 ticks)

- If `r25` is not `120`
  - If `r26` is not `252` (42 * 6 = 252)
  - Increment `r26` by 42
  - Set `OCR3BL` to `r26`
- Else

    - If `r26` is not `0`
    - Decrement `r26` by 42
    - Set `OCR3BL` to `r26`
- Return

ISR:[`Timer0OVF`]

- Disable global interrupts
- Increment register `r16` - Tick count
- Execute macro: `stopDebouncer` if `r16` is `156` (20ms has passed)
- Return from interrupt (+ enable global interrupts)

*156 ticks calculated by 10^6 microseconds / 128 microseconds per tick * 0.02 seconds (20 milliseconds)*

ISR:[`Timer1OVF`]

- Disable global interrupts
- Increment register `r22`
- Return from interrupt (+ enable global interrupts) if `r22` is not `651`
- Clear `r22`
- If macro: `isRunningISR` is true
    - Call function: `updateTurntable`
    - Increment `r23`
    - If `r23` is `12` ($\frac{1}{12}$ seconds * 12 = 1 second)
    - Clear `r23`
    - If `r21:r20` is 0
        - Call function: `doFinish`
    - Call function: `tickTime`
- Else
    - If `r25` is not `120` (10 seconds / $\frac{1}{12}$ = 120 ticks)
    - Increment `r25`
- Call function: `checkBacklight`
- Return from interrupt (+ enable global interrupts)

*651 ticks calculated by 10^6 microseconds / 128 microseconds per tick * $\frac{1}{12}$ seconds*

function:[`updateTurntable`]

- Increment `r24`
- Set `r24` to `0` if `r24` is `4`
- Load byte `turntablePositions+r24` into `r17`
- Set LCD Line 1 Column 16 to `r17`
- Return

[`init`] - Set up devices and ports

- Set up LCD (Using Lab 4 LCD example)
    - Reset display
    - Set 4-line mode
- Set up LCD backlight

    - Set `DDRL` register to `0b10000` - PL4 / OC5B for output
    - Set `TCCR5A` register to `0b00100001` - 8-bit phase-correct PWM, set on down-count
    - Set `TCCR5B` register to `0b1` - Clock to system clock (no prescaler)
    - Set `OCR5B` register to `0x00FF` - Current PWM duty cycle 100%
    - Set `r26` to `0xFF` (100% duty cycle)
- Set up keypad (Using Lab 4 Keypad example)
    - Set `DDRK` register to `0xF0` - K0-K3 for input; K4-K7 for output
    - Set `PCMSK2` register to `0xFF` - Enable PCINT23:16 triggers
    - Set `PCICR` register to `0b100` - Enable PCIE2 interrupt
    - Set `PORTK` register to `0x0F` - Enable all pins (allows interrupt to execute)
- Set up open and close buttons
    - Set `EICRA` register to `0b1010` - Falling edge for INT1 and INT0
    - Set `EIMSK` register to `0b11` - Enable INT1 and INT0
    - Set LCD Line 4 Column 16 to `'C'`
- Set up Turntable
    - Set `r24` to `3`
    - Call function: `updateTurntable` - Will set `r24` to `0` and display the horizontal position
- Set up Magnetron
    - Set `DDRE` register to `0b10000` - PE4 / OC3B for output
    - Set `OCR3B` register to `0x0000` - Current PWM duty cycle 0% (OFF)
    - Set `TCCR3A` register to `0b00100001` - 8-bit phase-correct PWM, set on down-count
    - Set `TCCR3B` register to `0b1` - Clock to system clock (no prescaler)
    - Set `r3` to `0xFF` - Set duty cycle to 100%
- Set up Magnetron level meter (LED Bar)
    - Set `DDRC` register to `0xFF` - Set all C ports as outputs
    - Set `r2` to `0xFF` - Show 8 LEDS (for 100%)
- Set up Door Open light (Strobe LED)
    - Bitwise OR `DDRG` with `0b100` - Set G2 as an output
- Set up debouncer
    - Set `TCCR0A` register to `0x0`
    - Set `TCCR0B` register to `0b10` - clock tap to clk_io/8
- Set up timer
    - Emulate 8-bit timer
    - Set `TCCR1B` register to `0b00001010` - enable CTC mode, and set clock tap to clk_io/8
    - Set `OCR1AL` register to `0xFF` (255)
    - Set `TIMSK1` to `0b1` - Start timer (will not activate turntable or countdown unless running)
- Set up program state
    - Set `r0` to `0b1` - Enter `ENTRY` state
    - Set `r1` to `1` - Enable button input
    - Clear `r21:r20` - Reset duration
    - Clear `r17` - Reset general purpose register
    - Clear `r18` - Reset general purpose register
- Enable global interrupts (`sei`)

[`main`] Main Loop

- *Idle and wait for interrupts…*

Improvements

Modularity

Each button function currently handles code in its own ISR, which may cause performance issues for a program that has time-critical operations to complete. For a microwave oven, not so much - but this idea can be incorporated.

As this implementation is event-driven (using interrupts as function triggers), the functionality of each button (depending on current state) could be passed into the main loop, which currently idles. The main loop could then be rewritten to handle button functionality.