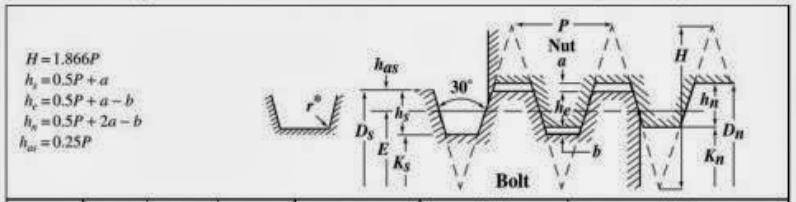
Lampiran

A. Lampiran Tabel

Trapezoidal Metric Thread — Preferred Basic Sizes (DIN 103)



Nom. & Major Diam.of Bolt, D_s	Pitch, P	Pitch Diam., E	Depth of Engage-ment, h_e	Clearance		Bolt		Nut		
				a	b	Minor Diam., K_s	Depth of Thread, h_s	Major Diam., D_n	Minor Diam., K_n	Depth of Thread, h_n
10	3	8.5	1.25	0.25	0.5	6.5	1.75	10.5	7.5	1.50
12	3	10.5	1.25	0.25	0.5	8.5	1.75	12.5	9.5	1.50
14	4	12	1.75	0.25	0.5	9.5	2.25	14.5	10.5	2.00
16	4	14	1.75	0.25	0.5	11.5	2.25	16.5	12.5	2.00
18	4	16	1.75	0.25	0.5	13.5	2.25	18.5	14.5	2.00
20	4	18	1.75	0.25	0.5	15.5	2.25	20.5	16.5	2.00
22	5	19.5	2	0.25	0.75	16.5	2.75	22.5	18	2.00
24	5	21.5	2	0.25	0.75	18.5	2.75	24.5	20	2.25
26	5	23.5	2	0.25	0.75	20.5	2.75	26.5	22	2.25
28	5	25.5	2	0.25	0.75	22.5	2.75	28.5	24	2.25
30	6	27	2.5	0.25	0.75	23.5	3.25	30.5	25	2.75
32	6	29	2.5	0.25	0.75	25.5	3.25	32.5	27	2.75
36	6	33	2.5	0.25	0.75	29.5	3.25	36.5	31	2.75
40	7	36.5	3	0.25	0.75	32.5	3.75	40.5	34	3.25
44	7	40.5	3	0.25	0.75	36.5	3.75	44.5	38	3.25
48	8	44	3.5	0.25	0.75	39.5	4.25	48.5	41	3.75
50	8	46	3.5	0.25	0.75	41.5	4.25	50.5	43	3.75
52	8	48	3.5	0.25	0.75	43.5	4.25	52.5	45	3.75
55	9	50.5	4	0.25	0.75	45.5	4.75	55.5	47	4.25
60	9	55.5	4	0.25	0.75	50.5	4.75	60.5	52	4.25
65	10	60	4.5	0.25	0.75	54.5	5.25	65.5	56	4.75
70	10	65	4.5	0.25	0.75	59.5	5.25	70.5	61	4.75
75	10	70	4.5	0.25	0.75	64.5	5.25	75.5	66	4.75
80	10	75	4.5	0.25	0.75	69.5	5.25	80.5	71	4.75
85	12	79	5.5	0.25	0.75	72.5	6.25	85.5	74	5.75
90	12	84	5.5	0.25	0.75	77.5	6.25	90.5	79	5.75
95	12	89	5.5	0.25	0.75	82.5	6.25	95.5	84	5.75
100	12	94	5.5	0.25	0.75	87.5	6.25	100.5	89	5.75
110	12	104	5.5	0.25	0.75	97.5	6.25	110.5	99	5.75
120	14	113	6	0.5	1.5	105	7.5	121	108	6.5
130	14	123	6	0.5	1.5	115	7.5	131	118	6.5
140	14	133	6	0.5	1.5	125	7.5	141	128	6.5
150	16	142	7	0.5	1.5	133	8.5	151	136	7.5
160	16	152	7	0.5	1.5	143	8.5	161	146	7.5
170	16	162	7	0.5	1.5	153	8.5	171	156	7.5
180	18	171	8	0.5	1.5	161	9.5	181	164	8.5
190	18	181	8	0.5	1.5	171	9.5	191	174	8.5
200	18	191	8	0.5	1.5	181	9.5	201	184	8.5
210	20	200	9	0.5	1.5	189	10.5	211	192	9.5
220	20	210	9	0.5	1.5	199	10.5	221	202	9.5
230	20	220	9	0.5	1.5	209	10.5	231	212	9.5
240	22	229	10	0.5	1.5	217	11.5	241	220	10.5
250	22	239	10	0.5	1.5	227	11.5	251	230	10.5
260	22	249	10	0.5	1.5	237	11.5	261	240	10.5
270	24	258	11	0.5	1.5	245	12.5	271	248	11.5
280	24	268	11	0.5	1.5	255	12.5	281	258	11.5
290	24	278	11	0.5	1.5	265	12.5	291	268	11.5
300	26	287	12	0.5	1.5	273	13.5	301	276	12.5

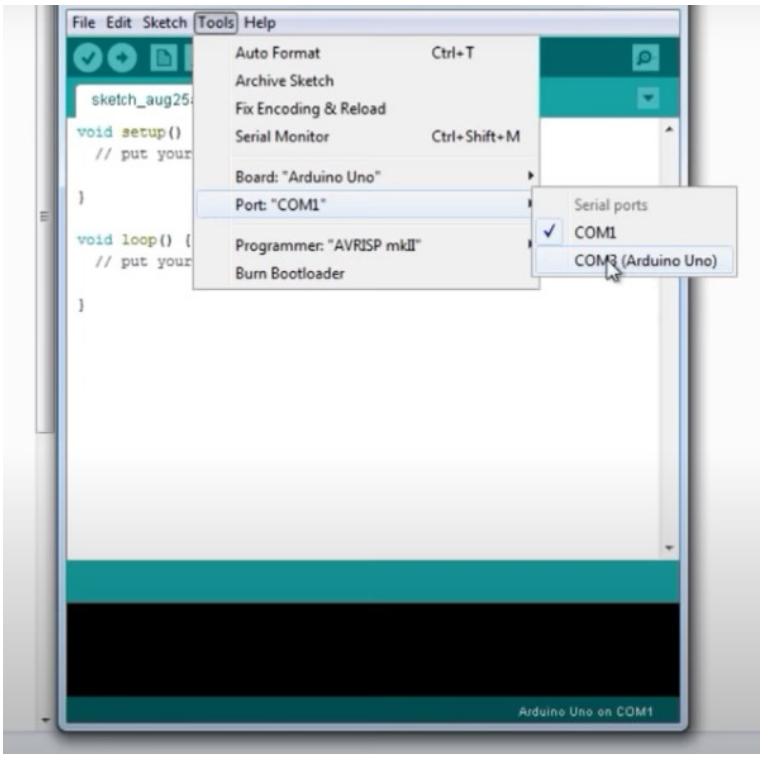
All dimensions are in millimeters.

*Roots are rounded to a radius, r, equal to 0.25 mm for pitches of from 3 to 12 mm inclusive and 0.5 mm for pitches of from 14 to 26 mm inclusive for power transmission.

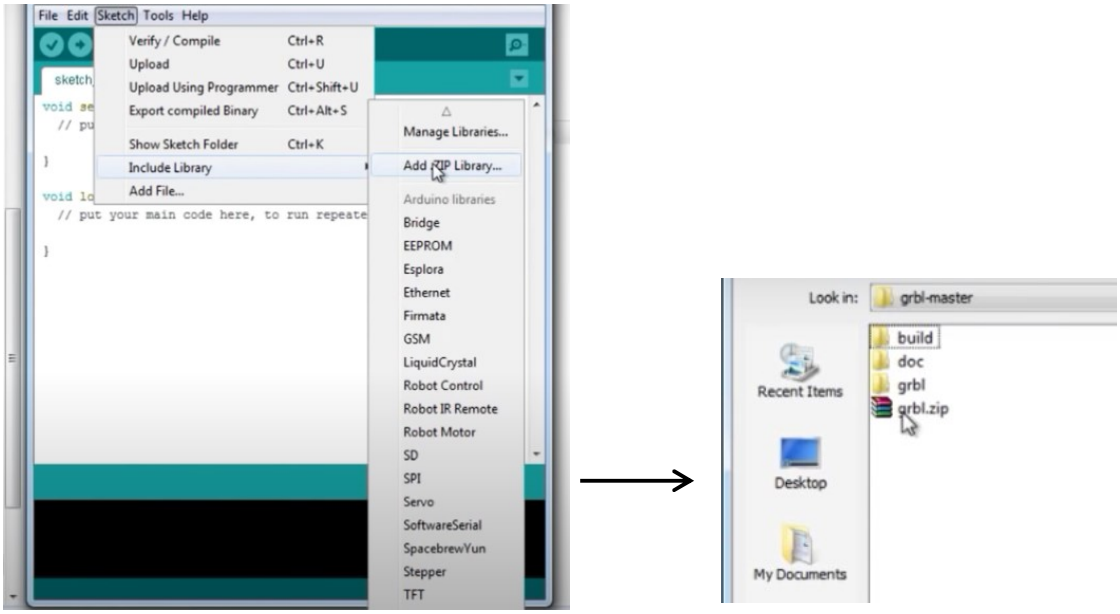
Tabel A.1 ulir trapesium

(Sumber:http://machiningtool.blogspot.com/2014/09/macam-macam-jenis-ulir-types-of-thread.html)

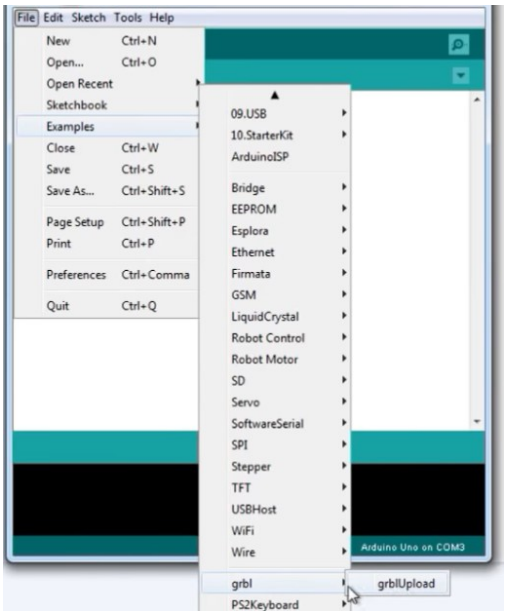
B. Lampiran Arduino



Gambar B.1 Memastikan *Port* pada USB dari arduino terdeteksi di laptop/PC



Gambar B.2 Memasukan Grbl kedalam apk arduino



Gambar B.3 Kemudian membuka *file* Grbl yang telah di tambahkan

Gambar B.4 Tampilan program Grbl (program tidak di ubah-ubah), lalu *upload*

C. Lampiran Pemrograman Grbl

Pemrograman C.1 *Config*

```
/*
   config.h - compile time configuration
   Part of Grbl

   Copyright (c) 2012-2015 Sungeun K. Jeon
   Copyright (c) 2009-2011 Simen Svale Skogsrud

   Grbl is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.

   Grbl is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

// This file contains compile-time configurations for Grbl's internal system. For the most part,
// users will not need to directly modify these, but they are here for specific needs, i.e.
// performance tuning or adjusting to non-typical machines.

// IMPORTANT: Any changes here requires a full re-compiling of the source code to propagate them.

#ifndef config_h
#define config_h
#include "grbl.h" // For Arduino IDE compatibility.


// Default settings. Used when resetting EEPROM. Change to desired name in defaults.h
#define DEFAULTS_GENERIC

// Serial baud rate
#define BAUD_RATE 115200

// Default cpu mappings. Grbl officially supports the Arduino Uno only. Other processor types
// may exist from user-supplied templates or directly user-defined in cpu_map.h
#define CPU_MAP_ATMEGA328P // Arduino Uno CPU

// Define realtime command special characters. These characters are 'picked-off' directly from the
// serial read data stream and are not passed to the grbl line execution parser. Select characters
// that do not and must not exist in the streamed g-code program. ASCII control characters may be
// used, if they are available per user setup. Also, extended ASCII codes (>127), which are never in
// g-code programs, maybe selected for interface programs.
// NOTE: If changed, manually update help message in report.c.
#define CMD_STATUS_REPORT '?'
#define CMD_FEED_HOLD '!'
#define CMD_CYCLE_START '~'
#define CMD_RESET 0x18 // ctrl-x.
#define CMD_SAFETY_DOOR '@'

// If homing is enabled, homing init lock sets Grbl into an alarm state upon power up. This forces
```

```
// the user to perform the homing cycle (or override the locks) before doing anything else. This is
// mainly a safety feature to remind the user to home, since position is unknown to Grbl.
#define HOMING_INIT_LOCK // Comment to disable

// Define the homing cycle patterns with bitmasks. The homing cycle first performs a search mode
// to quickly engage the limit switches, followed by a slower locate mode, and finished by a short
// pull-off motion to disengage the limit switches. The following HOMING_CYCLE_x defines are executed
// in order starting with suffix 0 and completes the homing routine for the specified-axes only. If
// an axis is omitted from the defines, it will not home, nor will the system update its position.
// Meaning that this allows for users with non-standard cartesian machines, such as a lathe (x then z,
// with no y), to configure the homing cycle behavior to their needs.
// NOTE: The homing cycle is designed to allow sharing of limit pins, if the axes are not in the same
// cycle, but this requires some pin settings changes in cpu_map.h file. For example, the default homing
// cycle can share the Z limit pin with either X or Y limit pins, since they are on different cycles.
// By sharing a pin, this frees up a precious IO pin for other purposes. In theory, all axes limit pins
// may be reduced to one pin, if all axes are homed with seperate cycles, or vice versa, all three axes
// on separate pin, but homed in one cycle. Also, it should be noted that the function of hard limits
// will not be affected by pin sharing.
// NOTE: Defaults are set for a traditional 3-axis CNC machine. Z-axis first to clear, followed by X & Y.
#define HOMING_CYCLE_0 (1<<Z_AXIS)                // REQUIRED: First move Z to clear workspace.
#define HOMING_CYCLE_1 ((1<<X_AXIS)|(1<<Y_AXIS))  // OPTIONAL: Then move X,Y at the same time.
// #define HOMING_CYCLE_2                         // OPTIONAL: Uncomment and add axes mask to enable

// Number of homing cycles performed after when the machine initially jogs to limit switches.
// This help in preventing overshoot and should improve repeatability. This value should be one or
// greater.
#define N_HOMING_LOCATE_CYCLE 1 // Integer (1-128)

// After homing, Grbl will set by default the entire machine space into negative space, as is typical
// for professional CNC machines, regardless of where the limit switches are located. Uncomment this
// define to force Grbl to always set the machine origin at the homed location despite switch orientation.
// #define HOMING_FORCE_SET_ORIGIN // Uncomment to enable.

// Number of blocks Grbl executes upon startup. These blocks are stored in EEPROM, where the size
// and addresses are defined in settings.h. With the current settings, up to 2 startup blocks may
// be stored and executed in order. These startup blocks would typically be used to set the g-code
// parser state depending on user preferences.
#define N_STARTUP_LINE 2 // Integer (1-2)

// Number of floating decimal points printed by Grbl for certain value types. These settings are
// determined by realistic and commonly observed values in CNC machines. For example, position
// values cannot be less than 0.001mm or 0.0001in, because machines can not be physically more
// precise this. So, there is likely no need to change these, but you can if you need to here.
// NOTE: Must be an integer value from 0 to ~4. More than 4 may exhibit round-off errors.
#define N_DECIMAL_COORDVALUE_INCH 4 // Coordinate or position value in inches
#define N_DECIMAL_COORDVALUE_MM   3 // Coordinate or position value in mm
#define N_DECIMAL_RATEVALUE_INCH  1 // Rate or velocity value in in/min
#define N_DECIMAL_RATEVALUE_MM    0 // Rate or velocity value in mm/min
#define N_DECIMAL_SETTINGVALUE    3 // Decimals for floating point setting values

// If your machine has two limits switches wired in parallel to one axis, you will need to enable
// this feature. Since the two switches are sharing a single pin, there is no way for Grbl to tell
// which one is enabled. This option only effects homing, where if a limit is engaged, Grbl will
// alarm out and force the user to manually disengage the limit switch. Otherwise, if you have one
```

```
// limit switch for each axis, don't enable this option. By keeping it disabled, you can perform a
// homing cycle while on the limit switch and not have to move the machine off of it.
// #define LIMITS_TWO_SWITCHES_ON_AXES

// Allows GRBL to track and report gcode line numbers.  Enabling this means that the planning buffer
// goes from 18 or 16 to make room for the additional line number data in the plan_block_t struct
  #define USE_LINE_NUMBERS // Disabled by default. Uncomment to enable.

// Allows GRBL to report the real-time feed rate.  Enabling this means that GRBL will be reporting
more
// data with each status update.
// NOTE: This is experimental and doesn't quite work 100%. Maybe fixed or refactored later.
// #define REPORT_REALTIME_RATE // Disabled by default. Uncomment to enable.

// Upon a successful probe cycle, this option provides immediately feedback of the probe coordinates
// through an automatically generated message. If disabled, users can still access the last probe
// coordinates through Grbl '$#' print parameters.
#define MESSAGE_PROBE_COORDINATES // Enabled by default. Comment to disable.

// Enables a second coolant control pin via the mist coolant g-code command M7 on the Arduino Uno
// analog pin 4. Only use this option if you require a second coolant control pin.
// NOTE: The M8 flood coolant control pin on analog pin 3 will still be functional regardless.
// #define ENABLE_M7 // Disabled by default. Uncomment to enable.

// This option causes the feed hold input to act as a safety door switch. A safety door, when triggered,
// immediately forces a feed hold and then safely de-energizes the machine. Resuming is blocked until
// the safety door is re-engaged. When it is, Grbl will re-energize the machine and then resume on the
// previous tool path, as if nothing happened.
// #define ENABLE_SAFETY_DOOR_INPUT_PIN // Default disabled. Uncomment to enable.

// After the safety door switch has been toggled and restored, this setting sets the power-up delay
// between restoring the spindle and coolant and resuming the cycle.
// NOTE: Delay value is defined in milliseconds from zero to 65,535.
#define SAFETY_DOOR_SPINDLE_DELAY 4000
#define SAFETY_DOOR_COOLANT_DELAY 1000

// Enable CoreXY kinematics. Use ONLY with CoreXY machines.
// IMPORTANT: If homing is enabled, you must reconfigure the homing cycle #defines above to
// #define HOMING_CYCLE_0 (1<<X_AXIS) and #define HOMING_CYCLE_1 (1<<Y_AXIS)
// NOTE: This configuration option alters the motion of the X and Y axes to principle of operation
// defined at (http://corexy.com/theory.html). Motors are assumed to positioned and wired exactly as
// described, if not, motions may move in strange directions. Grbl requires the CoreXY A and B motors
// have the same steps per mm internally.
// #define COREXY // Default disabled. Uncomment to enable.

// Inverts pin logic of the control command pins. This essentially means when this option is enabled
// you can use normally-closed switches, rather than the default normally-open switches.
// NOTE: If you require individual control pins inverted, keep this macro disabled and simply alter
//     the CONTROL_INVERT_MASK definition in cpu_map.h files.
// #define INVERT_ALL_CONTROL_PINS // Default disabled. Uncomment to enable.

// Inverts select limit pin states based on the following mask. This effects all limit pin functions,
// such as hard limits and homing. However, this is different from overall invert limits setting.
// This build option will invert only the limit pins defined here, and then the invert limits setting
// will be applied to all of them. This is useful when a user has a mixed set of limit pins with both
// normally-open(NO) and normally-closed(NC) switches installed on their machine.
// NOTE: PLEASE DO NOT USE THIS, unless you have a situation that needs it.
// #define INVERT_LIMIT_PIN_MASK ((1<<X_LIMIT_BIT)|(1<<Y_LIMIT_BIT)) // Default
disabled. Uncomment to enable.
```

```
// Inverts the spindle enable pin from low-disabled/high-enabled to low-enabled/high-disabled. Useful
// for some pre-built electronic boards.
// NOTE: If VARIABLE_SPINDLE is enabled(default), this option has no effect as the PWM output and
// spindle enable are combined to one pin. If you need both this option and spindle speed PWM,
// uncomment the config option USE_SPINDLE_DIR_AS_ENABLE_PIN below.
// #define INVERT_SPINDLE_ENABLE_PIN // Default disabled. Uncomment to enable.

// Enable control pin states feedback in status reports. The data is presented as simple binary of
// the control pin port (0 (low) or 1(high)), masked to show only the input pins. Non-control pins on the
// port will always show a 0 value. See cpu_map.h for the pin bitmap. As with the limit pin reporting,
// we do not recommend keeping this option enabled. Try to only use this for setting up a new CNC.
// #define REPORT_CONTROL_PIN_STATE // Default disabled. Uncomment to enable.

// When Grbl powers-cycles or is hard reset with the Arduino reset button, Grbl boots up with no ALARM
// by default. This is to make it as simple as possible for new users to start using Grbl. When homing
// is enabled and a user has installed limit switches, Grbl will boot up in an ALARM state to indicate
// Grbl doesn't know its position and to force the user to home before proceeding. This option forces
// Grbl to always initialize into an ALARM state regardless of homing or not. This option is more for
// OEMs and LinuxCNC users that would like this power-cycle behavior.
// #define FORCE_INITIALIZATION_ALARM // Default disabled. Uncomment to enable.

// ----------------------------------------------------------------------------------
// ADVANCED CONFIGURATION OPTIONS:

// Enables minimal reporting feedback mode for GUIs, where human-readable strings are not as important.
// This saves nearly 2KB of flash space and may allow enough space to install other/future features.
// GUIs will need to install a look-up table for the error-codes that Grbl sends back in their place.
// NOTE: This feature is new and experimental. Make sure the GUI you are using supports this mode.
// #define REPORT_GUI_MODE // Default disabled. Uncomment to enable.

// The temporal resolution of the acceleration management subsystem. A higher number gives smoother
// acceleration, particularly noticeable on machines that run at very high feedrates, but may negatively
// impact performance. The correct value for this parameter is machine dependent, so it's advised to
// set this only as high as needed. Approximate successful values can widely range from 50 to 200 or more.
// NOTE: Changing this value also changes the execution time of a segment in the step segment buffer.
// When increasing this value, this stores less overall time in the segment buffer and vice versa. Make
// certain the step segment buffer is increased/decreased to account for these changes.
#define ACCELERATION_TICKS_PER_SECOND 100

// Adaptive Multi-Axis Step Smoothing (AMASS) is an advanced feature that does what its name implies,
// smoothing the stepping of multi-axis motions. This feature smooths motion particularly at low step
// frequencies below 10kHz, where the aliasing between axes of multi-axis motions can cause audible
// noise and shake your machine. At even lower step frequencies, AMASS adapts and provides even better
// step smoothing. See stepper.c for more details on the AMASS system works.
#define ADAPTIVE_MULTI_AXIS_STEP_SMOOTHING    // Default enabled. Comment to disable.

// Sets the maximum step rate allowed to be written as a Grbl setting. This option enables an error
// check in the settings module to prevent settings values that will exceed this limitation. The maximum
// step rate is strictly limited by the CPU speed and will change if something other than an AVR running
// at 16MHz is used.
// NOTE: For now disabled, will enable if flash space permits.
// #define MAX_STEP_RATE_HZ 30000 // Hz
```

```
// By default, Grbl sets all input pins to normal-high operation with their internal pull-up resistors
// enabled. This simplifies the wiring for users by requiring only a switch connected to ground,
// although its recommended that users take the extra step of wiring in low-pass filter to reduce
// electrical noise detected by the pin. If the user inverts the pin in Grbl settings, this just flips
// which high or low reading indicates an active signal. In normal operation, this means the user
// needs to connect a normal-open switch, but if inverted, this means the user should connect a
// normal-closed switch.
// The following options disable the internal pull-up resistors, sets the pins to a normal-low
// operation, and switches must be now connect to Vcc instead of ground. This also flips the meaning
// of the invert pin Grbl setting, where an inverted setting now means the user should connect a
// normal-open switch and vice versa.
// NOTE: All pins associated with the feature are disabled, i.e. XYZ limit pins, not individual axes.
// WARNING: When the pull-ups are disabled, this requires additional wiring with pull-down resistors!
//#define DISABLE_LIMIT_PIN_PULL_UP
//#define DISABLE_PROBE_PIN_PULL_UP
//#define DISABLE_CONTROL_PIN_PULL_UP

// Sets which axis the tool length offset is applied. Assumes the spindle is always parallel with
// the selected axis with the tool oriented toward the negative direction. In other words, a positive
// tool length offset value is subtracted from the current location.
#define TOOL_LENGTH_OFFSET_AXIS Z_AXIS // Default z-axis. Valid values are X_AXIS, Y_AXIS, or Z_AXIS.

// Enables variable spindle output voltage for different RPM values. On the Arduino Uno, the spindle
// enable pin will output 5V for maximum RPM with 256 intermediate levels and 0V when disabled.
// NOTE: IMPORTANT for Arduino Unos! When enabled, the Z-limit pin D11 and spindle enable pin D12 switch!
// The hardware PWM output on pin D11 is required for variable spindle output voltages.
#define VARIABLE_SPINDLE // Default enabled. Comment to disable.

// Used by the variable spindle output only. These parameters set the maximum and minimum spindle speed
// "S" g-code values to correspond to the maximum and minimum pin voltages. There are 256 discrete and
// equally divided voltage bins between the maximum and minimum spindle speeds. So for a 5V pin, 1000
// max rpm, and 250 min rpm, the spindle output voltage would be set for the following "S" commands:
// "S1000" @ 5V, "S250" @ 0.02V, and "S625" @ 2.5V (mid-range). The pin outputs 0V when disabled.
#define SPINDLE_MAX_RPM 24000.0 // Max spindle RPM. This value is equal to 100% duty cycle on the PWM.
#define SPINDLE_MIN_RPM 0.0     // Min spindle RPM. This value is equal to (1/256) duty cycle on the PWM.

// Used by variable spindle output only. This forces the PWM output to a minimum duty cycle when enabled.
// When disabled, the PWM pin will still read 0V. Most users will not need this option, but it may be
// useful in certain scenarios. This setting does not update the minimum spindle RPM calculations. Any
// spindle RPM output lower than this value will be set to this value.
// #define MINIMUM_SPINDLE_PWM 5 // Default disabled. Uncomment to enable. Integer (0-255)

// By default on a 328p(Uno), Grbl combines the variable spindle PWM and the enable into one pin to help
// preserve I/O pins. For certain setups, these may need to be separate pins. This configure option uses
// the spindle direction pin(D13) as a separate spindle enable pin along with spindle speed PWM on pin D11.
// NOTE: This configure option only works with VARIABLE_SPINDLE enabled and a 328p processor (Uno).
// NOTE: With no direction pin, the spindle clockwise M4 g-code command will be removed. M3 and M5 still work.
```

```
// NOTE: BEWARE! The Arduino bootloader toggles the D13 pin when it powers up. If you flash Grbl with
// a programmer (you can use a spare Arduino as "Arduino as ISP". Search the web on how to wire this.),
// this D13 LED toggling should go away. We haven't tested this though. Please report how it goes!
// #define USE_SPINDLE_DIR_AS_ENABLE_PIN // Default disabled. Uncomment to enable.

// With this enabled, Grbl sends back an echo of the line it has received, which has been pre-parsed (spaces
// removed, capitalized letters, no comments) and is to be immediately executed by Grbl. Echoes will not be
// sent upon a line buffer overflow, but should for all normal lines sent to Grbl. For example, if a user
// sendss the line 'g1 x1.032 y2.45 (test comment)', Grbl will echo back in the form '[echo: G1X1.032Y2.45]'.
// NOTE: Only use this for debugging purposes!! When echoing, this takes up valuable resources and can effect
// performance. If absolutely needed for normal operation, the serial write buffer should be greatly increased
// to help minimize transmission waiting within the serial write protocol.
  #define REPORT_ECHO_LINE_RECEIVED // Default disabled. Uncomment to enable.

// Minimum planner junction speed. Sets the default minimum junction speed the planner plans to at
// every buffer block junction, except for starting from rest and end of the buffer, which are always
// zero. This value controls how fast the machine moves through junctions with no regard for acceleration
// limits or angle between neighboring block line move directions. This is useful for machines that can't
// tolerate the tool dwelling for a split second, i.e. 3d printers or laser cutters. If used, this value
// should not be much greater than zero or to the minimum value necessary for the machine to work.
#define MINIMUM_JUNCTION_SPEED 0.0 // (mm/min)

// Sets the minimum feed rate the planner will allow. Any value below it will be set to this minimum
// value. This also ensures that a planned motion always completes and accounts for any floating-point
// round-off errors. Although not recommended, a lower value than 1.0 mm/min will likely work in smaller
// machines, perhaps to 0.1mm/min, but your success may vary based on multiple factors.
#define MINIMUM_FEED_RATE 1.0 // (mm/min)

// Number of arc generation iterations by small angle approximation before exact arc trajectory
// correction with expensive sin() and cos() calcualtions. This parameter maybe decreased if there
// are issues with the accuracy of the arc generations, or increased if arc execution is getting
// bogged down by too many trig calculations.
#define N_ARC_CORRECTION 12 // Integer (1-255)

// The arc G2/3 g-code standard is problematic by definition. Radius-based arcs have horrible numerical
// errors when arc at semi-circles(pi) or full-circles(2*pi). Offset-based arcs are much more accurate
// but still have a problem when arcs are full-circles (2*pi). This define accounts for the floating
// point issues when offset-based arcs are commanded as full circles, but get interpreted as extremely
// small arcs with around machine epsilon (1.2e-7rad) due to numerical round-off and precision issues.
// This define value sets the machine epsilon cutoff to determine if the arc is a full-circle or not.
// NOTE: Be very careful when adjusting this value. It should always be greater than 1.2e-7 but not too
// much greater than this. The default setting should capture most, if not all, full arc error situations.
#define ARC_ANGULAR_TRAVEL_EPSILON 5E-7 // Float (radians)

// Time delay increments performed during a dwell. The default value is set at 50ms, which provides
// a maximum time delay of roughly 55 minutes, more than enough for most any application. Increasing
// this delay will increase the maximum dwell time linearly, but also reduces the responsiveness of
// run-time command executions, like status reports, since these are performed between each dwell
// time step. Also, keep in mind that the Arduino delay timer is not very accurate for long delays.
#define DWELL_TIME_STEP 50 // Integer (1-255) (milliseconds)
```

```
// Creates a delay between the direction pin setting and corresponding step pulse by creating
// another interrupt (Timer2 compare) to manage it. The main Grbl interrupt (Timer1 compare)
// sets the direction pins, and does not immediately set the stepper pins, as it would in
// normal operation. The Timer2 compare fires next to set the stepper pins after the step
// pulse delay time, and Timer2 overflow will complete the step pulse, except now delayed
// by the step pulse time plus the step pulse delay. (Thanks langwadt for the idea!)
// NOTE: Uncomment to enable. The recommended delay must be > 3us, and, when added with the
// user-supplied step pulse time, the total time must not exceed 127us. Reported successful
// values for certain setups have ranged from 5 to 20us.
// #define STEP_PULSE_DELAY 10 // Step pulse delay in microseconds. Default disabled.

// The number of linear motions in the planner buffer to be planned at any give time. The vast
// majority of RAM that Grbl uses is based on this buffer size. Only increase if there is extra
// available RAM, like when re-compiling for a Mega or Sanguino. Or decrease if the Arduino
// begins to crash due to the lack of available RAM or if the CPU is having trouble keeping
// up with planning new incoming motions as they are executed.
// #define BLOCK_BUFFER_SIZE 18    // Uncomment to override default in planner.h.

// Governs the size of the intermediary step segment buffer between the step execution algorithm
// and the planner blocks. Each segment is set of steps executed at a constant velocity over a
// fixed time defined by ACCELERATION_TICKS_PER_SECOND. They are computed such that the
planner
// block velocity profile is traced exactly. The size of this buffer governs how much step
// execution lead time there is for other Grbl processes have to compute and do their thing
// before having to come back and refill this buffer, currently at ~50msec of step moves.
// #define SEGMENT_BUFFER_SIZE 6 // Uncomment to override default in stepper.h.

// Line buffer size from the serial input stream to be executed. Also, governs the size of
// each of the startup blocks, as they are each stored as a string of this size. Make sure
// to account for the available EEPROM at the defined memory address in settings.h and for
// the number of desired startup blocks.
// NOTE: 80 characters is not a problem except for extreme cases, but the line buffer size
// can be too small and g-code blocks can get truncated. Officially, the g-code standards
// support up to 256 characters. In future versions, this default will be increased, when
// we know how much extra memory space we can re-invest into this.
// #define LINE_BUFFER_SIZE 80    // Uncomment to override default in protocol.h

// Serial send and receive buffer size. The receive buffer is often used as another streaming
// buffer to store incoming blocks to be processed by Grbl when its ready. Most streaming
// interfaces will character count and track each block send to each block response. So,
// increase the receive buffer if a deeper receive buffer is needed for streaming and avaiable
// memory allows. The send buffer primarily handles messages in Grbl. Only increase if large
// messages are sent and Grbl begins to stall, waiting to send the rest of the message.
// NOTE: Buffer size values must be greater than zero and less than 256.
// #define RX_BUFFER_SIZE 128 // Uncomment to override defaults in serial.h
// #define TX_BUFFER_SIZE 64

// Toggles XON/XOFF software flow control for serial communications. Not officially supported
// due to problems involving the Atmega8U2 USB-to-serial chips on current Arduinos. The firmware
// on these chips do not support XON/XOFF flow control characters and the intermediate buffer
// in the chips cause latency and overflow problems with standard terminal programs. However,
// using specifically-programmed UI's to manage this latency problem has been confirmed to work.
// As well as, older FTDI FT232RL-based Arduinos(Duemilanove) are known to work with standard
// terminal programs since their firmware correctly manage these XON/XOFF characters. In any
// case, please report any successes to grbl administrators!
// #define ENABLE_XONXOFF // Default disabled. Uncomment to enable.

// A simple software debouncing feature for hard limit switches. When enabled, the interrupt
// monitoring the hard limit switch pins will enable the Arduino's watchdog timer to re-check
```

```
// the limit pin state after a delay of about 32msec. This can help with CNC machines with
// problematic false triggering of their hard limit switches, but it WILL NOT fix issues with
// electrical interference on the signal cables from external sources. It's recommended to first
// use shielded signal cables with their shielding connected to ground (old USB/computer cables
// work well and are cheap to find) and wire in a low-pass circuit into each limit pin.
// #define ENABLE_SOFTWARE_DEBOUNCE // Default disabled. Uncomment to enable.

// Force Grbl to check the state of the hard limit switches when the processor detects a pin
// change inside the hard limit ISR routine. By default, Grbl will trigger the hard limits
// alarm upon any pin change, since bouncing switches can cause a state check like this to
// misread the pin. When hard limits are triggered, they should be 100% reliable, which is the
// reason that this option is disabled by default. Only if your system/electronics can guarantee
// that the switches don't bounce, we recommend enabling this option. This will help prevent
// triggering a hard limit when the machine disengages from the switch.
// NOTE: This option has no effect if SOFTWARE_DEBOUNCE is enabled.
// #define HARD_LIMIT_FORCE_STATE_CHECK // Default disabled. Uncomment to enable.


// ---------------------------------------------------------------------------------
// COMPILE-TIME ERROR CHECKING OF DEFINE VALUES:

#ifndef HOMING_CYCLE_0
  #error "Required HOMING_CYCLE_0 not defined."
#endif

#if defined(USE_SPINDLE_DIR_AS_ENABLE_PIN) && !defined(VARIABLE_SPINDLE)
  #error "USE_SPINDLE_DIR_AS_ENABLE_PIN may only be used with VARIABLE_SPINDLE
enabled"
#endif

#if defined(USE_SPINDLE_DIR_AS_ENABLE_PIN) && !defined(CPU_MAP_ATMEGA328P)
  #error "USE_SPINDLE_DIR_AS_ENABLE_PIN may only be used with a 328p processor"
#endif

// ---------------------------------------------------------------------------------


#endif
```

Pemrograman C.2 *Coolant Control*

```
/*
  coolant_control.h - spindle control methods
  Part of Grbl

  Copyright (c) 2012-2015 Sungeun K. Jeon

  Grbl is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  Grbl is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
```

```
*/

#ifndef coolant_control_h
#define coolant_control_h


void coolant_init();
void coolant_stop();
void coolant_set_state(uint8_t mode);
void coolant_run(uint8_t mode);

#endif
```

Pemrograman C.3 CPU *Map*

```
/*
   cpu_map.h - CPU and pin mapping configuration file
   Part of Grbl

   Copyright (c) 2012-2015 Sungeun K. Jeon

   Grbl is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.

   Grbl is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

/* The cpu_map.h files serve as a central pin mapping selection file for different processor
     types, i.e. AVR 328p or AVR Mega 2560. Each processor has its own pin mapping file.
     (i.e. cpu_map_atmega328p.h)  Grbl officially supports the Arduino Uno, but the
     other supplied pin mappings are supplied by users, so your results may vary. */

// NOTE: With new processors, only add the define name and filename to use.

#ifndef cpu_map_h
#define cpu_map_h


#ifdef CPU_MAP_ATMEGA328P // (Arduino Uno) Officially supported by Grbl.
  #include "cpu_map/cpu_map_atmega328p.h"
#endif

#ifdef CPU_MAP_ATMEGA2560 // (Arduino Mega 2560) Working @EliteEng
  #include "cpu_map/cpu_map_atmega2560.h"
#endif

/*
#ifdef CPU_MAP_CUSTOM_PROC
  // For a custom pin map or different processor, copy and edit one of the available cpu
  // map files and modify it to your needs. Make sure the defined name is also changed in
  // the config.h file.
```

```
#endif
*/
#endif
```

Pemrograman C.4 *Defaults*

```
/*
  defaults.h - defaults settings configuration file
  Part of Grbl

  Copyright (c) 2012-2015 Sungeun K. Jeon

  Grbl is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  Grbl is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

/* The defaults.h file serves as a central default settings selector for different machine
    types, from DIY CNC mills to CNC conversions of off-the-shelf machines. The settings
    files listed here are supplied by users, so your results may vary. However, this should
    give you a good starting point as you get to know your machine and tweak the settings for
    your nefarious needs.
    Ensure one and only one of these DEFAULTS_XXX values is defined in config.h */

#ifndef defaults_h

// Only define the DEFAULT_XXX with where to find the corresponding default_XXX.h file.
// Don't #define defaults_h here, let the selected file do it. Prevents including more than one.

#ifdef DEFAULTS_GENERIC
  // Grbl generic default settings. Should work across different machines.
  #include "defaults/defaults_generic.h"
#endif

#ifdef DEFAULTS_SHERLINE_5400
  // Description: Sherline 5400 mill with three NEMA 23 Keling  KL23H256-21-8B 185 oz-in stepper motors,
  // driven by three Pololu A4988 stepper drivers with a 30V, 6A power supply at 1.5A per winding.
  #include "defaults/defaults_sherline.h"
#endif

#ifdef DEFAULTS_SHAPEOKO
  // Description: Shapeoko CNC mill with three NEMA 17 stepper motors, driven by Synthetos
  // grblShield with a 24V, 4.2A power supply.
  #include "defaults/defaults_shapeoko.h"
#endif

#ifdef DEFAULTS_SHAPEOKO_2
  // Description: Shapeoko CNC mill with three NEMA 17 stepper motors, driven by Synthetos
  // grblShield at 28V.
```

```
  #include "defaults/defaults_shapeoko2.h"
#endif

#ifdef DEFAULTS_SHAPEOKO_3
  // Description: Shapeoko CNC mill with three NEMA 23 stepper motors, driven by CarbideMotion
  #include "defaults/defaults_shapeoko3.h"
#endif

#ifdef DEFAULTS_X_CARVE_500MM
  // Description: X-Carve 3D Carver CNC mill with three 200 step/rev motors driven by Synthetos
  // grblShield at 24V.
  #include "defaults/defaults_x_carve_500mm.h"
#endif

#ifdef DEFAULTS_X_CARVE_1000MM
  // Description: X-Carve 3D Carver CNC mill with three 200 step/rev motors driven by Synthetos
  // grblShield at 24V.
  #include "defaults/defaults_x_carve_1000mm.h"
#endif

#ifdef DEFAULTS_ZEN_TOOLWORKS_7x7
  // Description: Zen Toolworks 7x7 mill with three Shinano SST43D2121 65oz-in NEMA 17 stepper
motors.
  // Leadscrew is different from some ZTW kits, where most are 1.25mm/rev rather than 8.0mm/rev
here.
  // Driven by 30V, 6A power supply and TI DRV8811 stepper motor drivers.
  #include "defaults/defaults_zen_toolworks_7x7.h"
#endif

#ifdef DEFAULTS_OXCNC
  // Grbl settings for OpenBuilds OX CNC Machine
  // http://www.openbuilds.com/builds/openbuilds-ox-cnc-machine.341/
  #include "defaults/defaults_oxcnc.h"
#endif

#ifdef DEFAULTS_SIMULATOR
  // Settings only for Grbl Simulator (www.github.com/grbl/grbl-sim)
  #include "defaults/defaults_simulator.h"
#endif

#endif
```

Pemrograman C.5 *Eeprom*

```
/*
  eeprom.h - EEPROM methods
  Part of Grbl

  Copyright (c) 2009-2011 Simen Svale Skogsrud

  Grbl is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  Grbl is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.
```

```
   You should have received a copy of the GNU General Public License
   along with Grbl.    If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef eeprom_h
#define eeprom_h

unsigned char eeprom_get_char(unsigned int addr);
void eeprom_put_char(unsigned int addr, unsigned char new_value);
void memcpy_to_eeprom_with_checksum(unsigned int destination, char *source, unsigned int size);
int memcpy_from_eeprom_with_checksum(char *destination, unsigned int source, unsigned int size);

#endif
```

Pemrograman C.6 G-code

```
/*
   gcode.h - rs274/ngc parser.
   Part of Grbl

   Copyright (c) 2011-2015 Sungeun K. Jeon
   Copyright (c) 2009-2011 Simen Svale Skogsrud

   Grbl is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.

   Grbl is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.    See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with Grbl.    If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef gcode_h
#define gcode_h


// Define modal group internal numbers for checking multiple command violations and tracking the
// type of command that is called in the block. A modal group is a group of g-code commands that are
// mutually exclusive, or cannot exist on the same line, because they each toggle a state or execute
// a unique motion. These are defined in the NIST RS274-NGC v3 g-code standard, available online,
// and are similar/identical to other g-code interpreters by manufacturers (Haas,Fanuc,Mazak,etc).
// NOTE: Modal group define values must be sequential and starting from zero.
#define MODAL_GROUP_G0 0 // [G4,G10,G28,G28.1,G30,G30.1,G53,G92,G92.1] Non-modal
#define MODAL_GROUP_G1 1 // [G0,G1,G2,G3,G38.2,G38.3,G38.4,G38.5,G80] Motion
#define MODAL_GROUP_G2 2 // [G17,G18,G19] Plane selection
#define MODAL_GROUP_G3 3 // [G90,G91] Distance mode
#define MODAL_GROUP_G4 4 // [G91.1] Arc IJK distance mode
#define MODAL_GROUP_G5 5 // [G93,G94] Feed rate mode
#define MODAL_GROUP_G6 6 // [G20,G21] Units
#define MODAL_GROUP_G7 7 // [G40] Cutter radius compensation mode. G41/42 NOT
SUPPORTED.
#define MODAL_GROUP_G8 8 // [G43.1,G49] Tool length offset
#define MODAL_GROUP_G12 9 // [G54,G55,G56,G57,G58,G59] Coordinate system selection
```

```
#define MODAL_GROUP_G13 10 // [G61] Control mode

#define MODAL_GROUP_M4 11    // [M0,M1,M2,M30] Stopping
#define MODAL_GROUP_M7 12 // [M3,M4,M5] Spindle turning
#define MODAL_GROUP_M8 13 // [M7,M8,M9] Coolant control

// #define OTHER_INPUT_F 14
// #define OTHER_INPUT_S 15
// #define OTHER_INPUT_T 16

// Define command actions for within execution-type modal groups (motion, stopping, non-modal).
Used
// internally by the parser to know which command to execute.

// Modal Group G0: Non-modal actions
#define NON_MODAL_NO_ACTION 0 // (Default: Must be zero)
#define NON_MODAL_DWELL 1 // G4
#define NON_MODAL_SET_COORDINATE_DATA 2 // G10
#define NON_MODAL_GO_HOME_0 3 // G28
#define NON_MODAL_SET_HOME_0 4 // G28.1
#define NON_MODAL_GO_HOME_1 5 // G30
#define NON_MODAL_SET_HOME_1 6 // G30.1
#define NON_MODAL_ABSOLUTE_OVERRIDE 7 // G53
#define NON_MODAL_SET_COORDINATE_OFFSET 8 // G92
#define NON_MODAL_RESET_COORDINATE_OFFSET 9 //G92.1

// Modal Group G1: Motion modes
#define MOTION_MODE_SEEK 0 // G0 (Default: Must be zero)
#define MOTION_MODE_LINEAR 1 // G1
#define MOTION_MODE_CW_ARC 2    // G2
#define MOTION_MODE_CCW_ARC 3    // G3
#define MOTION_MODE_PROBE_TOWARD 4 // G38.2 NOTE: G38.2, G38.3, G38.4, G38.5 must
be sequential. See report_gcode_modes().
#define MOTION_MODE_PROBE_TOWARD_NO_ERROR 5 // G38.3
#define MOTION_MODE_PROBE_AWAY 6 // G38.4
#define MOTION_MODE_PROBE_AWAY_NO_ERROR 7 // G38.5
#define MOTION_MODE_NONE 8 // G80

// Modal Group G2: Plane select
#define PLANE_SELECT_XY 0 // G17 (Default: Must be zero)
#define PLANE_SELECT_ZX 1 // G18
#define PLANE_SELECT_YZ 2 // G19

// Modal Group G3: Distance mode
#define DISTANCE_MODE_ABSOLUTE 0 // G90 (Default: Must be zero)
#define DISTANCE_MODE_INCREMENTAL 1 // G91

// Modal Group G4: Arc IJK distance mode
#define DISTANCE_ARC_MODE_INCREMENTAL 0 // G91.1 (Default: Must be zero)

// Modal Group M4: Program flow
#define PROGRAM_FLOW_RUNNING 0 // (Default: Must be zero)
#define PROGRAM_FLOW_PAUSED 1 // M0, M1
#define PROGRAM_FLOW_COMPLETED 2 // M2, M30

// Modal Group G5: Feed rate mode
#define FEED_RATE_MODE_UNITS_PER_MIN 0 // G94 (Default: Must be zero)
#define FEED_RATE_MODE_INVERSE_TIME 1 // G93

// Modal Group G6: Units mode
```

```
#define UNITS_MODE_MM 0 // G21 (Default: Must be zero)
#define UNITS_MODE_INCHES 1 // G20

// Modal Group G7: Cutter radius compensation mode
#define CUTTER_COMP_DISABLE 0 // G40 (Default: Must be zero)

// Modal Group G13: Control mode
#define CONTROL_MODE_EXACT_PATH 0 // G61 (Default: Must be zero)

// Modal Group M7: Spindle control
#define SPINDLE_DISABLE 0 // M5 (Default: Must be zero)
#define SPINDLE_ENABLE_CW 1 // M3
#define SPINDLE_ENABLE_CCW 2 // M4

// Modal Group M8: Coolant control
#define COOLANT_DISABLE 0 // M9 (Default: Must be zero)
#define COOLANT_MIST_ENABLE 1 // M7
#define COOLANT_FLOOD_ENABLE 2 // M8

// Modal Group G8: Tool length offset
#define TOOL_LENGTH_OFFSET_CANCEL 0 // G49 (Default: Must be zero)
#define TOOL_LENGTH_OFFSET_ENABLE_DYNAMIC 1 // G43.1

// Modal Group G12: Active work coordinate system
// N/A: Stores coordinate system value (54-59) to change to.


// Define parameter word mapping.
#define WORD_F    0
#define WORD_I    1
#define WORD_J    2
#define WORD_K    3
#define WORD_L    4
#define WORD_N    5
#define WORD_P    6
#define WORD_R    7
#define WORD_S    8
#define WORD_T    9
#define WORD_X    10
#define WORD_Y    11
#define WORD_Z    12


// NOTE: When this struct is zeroed, the above defines set the defaults for the system.
typedef struct {
   uint8_t motion;                // {G0,G1,G2,G3,G38.2,G80}
   uint8_t feed_rate;             // {G93,G94}
   uint8_t units;                 // {G20,G21}
   uint8_t distance;              // {G90,G91}
   // uint8_t distance_arc; // {G91.1} NOTE: Don't track. Only default supported.
   uint8_t plane_select;       // {G17,G18,G19}
   // uint8_t cutter_comp;    // {G40} NOTE: Don't track. Only default supported.
   uint8_t tool_length;         // {G43.1,G49}
   uint8_t coord_select;       // {G54,G55,G56,G57,G58,G59}
   // uint8_t control;           // {G61} NOTE: Don't track. Only default supported.
   uint8_t program_flow;       // {M0,M1,M2,M30}
   uint8_t coolant;               // {M7,M8,M9}
   uint8_t spindle;               // {M3,M4,M5}
} gc_modal_t;
```

```
typedef struct {
   float f;               // Feed
   float ijk[3];       // I,J,K Axis arc offsets
   uint8_t l;             // G10 or canned cycles parameters
   int32_t n;             // Line number
   float p;               // G10 or dwell parameters
   // float q;          // G82 peck drilling
   float r;               // Arc radius
   float s;               // Spindle speed
   uint8_t t;             // Tool selection
   float xyz[3];       // X,Y,Z Translational axes
} gc_values_t;


typedef struct {
   gc_modal_t modal;

   float spindle_speed;                // RPM
   float feed_rate;                    // Millimeters/min
   uint8_t tool;                        // Tracks tool number. NOT USED.
   int32_t line_number;                // Last line number sent

   float position[N_AXIS];             // Where the interpreter considers the tool to be at this point in the
code

   float coord_system[N_AXIS];      // Current work coordinate system (G54+). Stores offset from
absolute machine
                                       // position in mm. Loaded from EEPROM when called.
   float coord_offset[N_AXIS];      // Retains the G92 coordinate offset (work coordinates) relative to
                                       // machine zero in mm. Non-persistent. Cleared upon reset and
boot.
   float tool_length_offset;         // Tracks tool length offset value when enabled.
} parser_state_t;
extern parser_state_t gc_state;

typedef struct {
//      uint16_t command_words;    // NOTE: If this bitflag variable fills, G and M words can be
separated.
//      uint16_t value_words;

   uint8_t non_modal_command;
   gc_modal_t modal;
   gc_values_t values;

} parser_block_t;
extern parser_block_t gc_block;

// Initialize the parser
void gc_init();

// Execute one block of rs275/ngc/g-code
uint8_t gc_execute_line(char *line);

// Set g-code parser position. Input in steps.
void gc_sync_position();

#endif
```

Pemrograman C.7 Grbl

```
/*
  grbl.h - main Grbl include file
  Part of Grbl

  Copyright (c) 2015 Sungeun K. Jeon

  Grbl is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  Grbl is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef grbl_h
#define grbl_h

// Grbl versioning system
#define GRBL_VERSION "0.9j"
#define GRBL_VERSION_BUILD "20160726"

// Define standard libraries used by Grbl.
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>
#include <util/delay.h>
#include <math.h>
#include <inttypes.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>

// Define the Grbl system include files. NOTE: Do not alter organization.
#include "config.h"
#include "nuts_bolts.h"
#include "settings.h"
#include "system.h"
#include "defaults.h"
#include "cpu_map.h"
#include "coolant_control.h"
#include "eeprom.h"
#include "gcode.h"
#include "limits.h"
#include "motion_control.h"
#include "planner.h"
#include "print.h"
#include "probe.h"
#include "protocol.h"
#include "report.h"
#include "serial.h"
```

```
#include "spindle_control.h"
#include "stepper.h"

#endif
```

Pemrograman C.8 *Limits*

```
/*
  limits.h - code pertaining to limit-switches and performing the homing cycle
  Part of Grbl

  Copyright (c) 2012-2015 Sungeun K. Jeon
  Copyright (c) 2009-2011 Simen Svale Skogsrud

  Grbl is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  Grbl is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef limits_h
#define limits_h


// Initialize the limits module
void limits_init();

// Disables hard limits.
void limits_disable();

// Returns limit state as a bit-wise uint8 variable.
uint8_t limits_get_state();

// Perform one portion of the homing cycle based on the input settings.
void limits_go_home(uint8_t cycle_mask);

// Check for soft limit violations
void limits_soft_check(float *target);

#endif
```

Pemrograman C.9 *Motion Control*

```
/*
  motion_control.h - high level interface for issuing motion commands
  Part of Grbl

  Copyright (c) 2011-2015 Sungeun K. Jeon
  Copyright (c) 2009-2011 Simen Svale Skogsrud

  Grbl is free software: you can redistribute it and/or modify
```

```
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.

   Grbl is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef motion_control_h
#define motion_control_h


#define HOMING_CYCLE_LINE_NUMBER -1

// Execute linear motion in absolute millimeter coordinates. Feed rate given in millimeters/second
// unless invert_feed_rate is true. Then the feed_rate means that the motion should be completed in
// (1 minute)/feed_rate time.
#ifdef USE_LINE_NUMBERS
void mc_line(float *target, float feed_rate, uint8_t invert_feed_rate, int32_t line_number);
#else
void mc_line(float *target, float feed_rate, uint8_t invert_feed_rate);
#endif

// Execute an arc in offset mode format. position == current xyz, target == target xyz,
// offset == offset from current xyz, axis_XXX defines circle plane in tool space, axis_linear is
// the direction of helical travel, radius == circle radius, is_clockwise_arc boolean. Used
// for vector transformation direction.
#ifdef USE_LINE_NUMBERS
void mc_arc(float *position, float *target, float *offset, float radius, float feed_rate,
   uint8_t invert_feed_rate, uint8_t axis_0, uint8_t axis_1, uint8_t axis_linear, uint8_t
is_clockwise_arc, int32_t line_number);
#else
void mc_arc(float *position, float *target, float *offset, float radius, float feed_rate,
   uint8_t invert_feed_rate, uint8_t axis_0, uint8_t axis_1, uint8_t axis_linear, uint8_t
is_clockwise_arc);
#endif

// Dwell for a specific number of seconds
void mc_dwell(float seconds);

// Perform homing cycle to locate machine zero. Requires limit switches.
void mc_homing_cycle();

// Perform tool length probe cycle. Requires probe switch.
#ifdef USE_LINE_NUMBERS
void mc_probe_cycle(float *target, float feed_rate, uint8_t invert_feed_rate, uint8_t is_probe_away,
   uint8_t is_no_error, int32_t line_number);
#else
void mc_probe_cycle(float *target, float feed_rate, uint8_t invert_feed_rate, uint8_t is_probe_away,
   uint8_t is_no_error);
#endif

// Performs system reset. If in motion state, kills all motion and sets system alarm.
void mc_reset();
```

```
#endif
```

Pemrograman C.10 *Nuts Bolts*

```
/*
  nuts_bolts.h - Header file for shared definitions, variables, and functions
  Part of Grbl

  Copyright (c) 2011-2015 Sungeun K. Jeon
  Copyright (c) 2009-2011 Simen Svale Skogsrud

  Grbl is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  Grbl is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef nuts_bolts_h
#define nuts_bolts_h

#define false 0
#define true 1

// Axis array index values. Must start with 0 and be continuous.
#define N_AXIS 3 // Number of axes
#define X_AXIS 0 // Axis indexing value.
#define Y_AXIS 1
#define Z_AXIS 2
// #define A_AXIS 3

// CoreXY motor assignments. DO NOT ALTER.
// NOTE: If the A and B motor axis bindings are changed, this effects the CoreXY equations.
#ifdef COREXY
  #define A_MOTOR X_AXIS // Must be X_AXIS
  #define B_MOTOR Y_AXIS // Must be Y_AXIS
#endif

// Conversions
#define MM_PER_INCH (25.40)
#define INCH_PER_MM (0.0393701)
#define TICKS_PER_MICROSECOND (F_CPU/1000000)

// Useful macros
#define clear_vector(a) memset(a, 0, sizeof(a))
#define clear_vector_float(a) memset(a, 0.0, sizeof(float)*N_AXIS)
// #define clear_vector_long(a) memset(a, 0.0, sizeof(long)*N_AXIS)
#define max(a,b) (((a) > (b)) ? (a) : (b))
#define min(a,b) (((a) < (b)) ? (a) : (b))

// Bit field and masking macros
#define bit(n) (1 << n)
```

```
#define bit_true_atomic(x,mask) {uint8_t sreg = SREG; cli(); (x) |= (mask); SREG = sreg; }
#define bit_false_atomic(x,mask) {uint8_t sreg = SREG; cli(); (x) &= ~(mask); SREG = sreg; }
#define bit_toggle_atomic(x,mask) {uint8_t sreg = SREG; cli(); (x) ^= (mask); SREG = sreg; }
#define bit_true(x,mask) (x) |= (mask)
#define bit_false(x,mask) (x) &= ~(mask)
#define bit_istrue(x,mask) ((x & mask) != 0)
#define bit_isfalse(x,mask) ((x & mask) == 0)

// Read a floating point value from a string. Line points to the input buffer, char_counter
// is the indexer pointing to the current character of the line, while float_ptr is
// a pointer to the result variable. Returns true when it succeeds
uint8_t read_float(char *line, uint8_t *char_counter, float *float_ptr);

// Delays variable-defined milliseconds. Compiler compatibility fix for _delay_ms().
void delay_ms(uint16_t ms);

// Delays variable-defined microseconds. Compiler compatibility fix for _delay_us().
void delay_us(uint32_t us);

// Computes hypotenuse, avoiding avr-gcc's bloated version and the extra error checking.
float hypot_f(float x, float y);

#endif
```

Pemrograman C.11 *Planner*

```
/*
  planner.h - buffers movement commands and manages the acceleration profile plan
  Part of Grbl

  Copyright (c) 2011-2015 Sungeun K. Jeon
  Copyright (c) 2009-2011 Simen Svale Skogsrud

  Grbl is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  Grbl is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef planner_h
#define planner_h


// The number of linear motions that can be in the plan at any give time
#ifndef BLOCK_BUFFER_SIZE
  #ifdef USE_LINE_NUMBERS
    #define BLOCK_BUFFER_SIZE 16
  #else
    #define BLOCK_BUFFER_SIZE 18
  #endif
#endif
```

```
// This struct stores a linear movement of a g-code block motion with its critical "nominal" values
// are as specified in the source g-code.
typedef struct {
  // Fields used by the bresenham algorithm for tracing the line
  // NOTE: Used by stepper algorithm to execute the block correctly. Do not alter these values.
  uint8_t direction_bits;        // The direction bit set for this block (refers to *_DIRECTION_BIT in config.h)
  uint32_t steps[N_AXIS];        // Step count along each axis
  uint32_t step_event_count; // The maximum step axis count and number of steps required to complete this block.

  // Fields used by the motion planner to manage acceleration
  float entry_speed_sqr;              // The current planned entry speed at block junction in (mm/min)^2
  float max_entry_speed_sqr;          // Maximum allowable entry speed based on the minimum of junction limit and
                                      //          neighboring nominal speeds with overrides in (mm/min)^2
  float max_junction_speed_sqr;      // Junction entry speed limit based on direction vectors in (mm/min)^2
  float nominal_speed_sqr;           // Axis-limit adjusted nominal speed for this block in (mm/min)^2
  float acceleration;                // Axis-limit adjusted line acceleration in (mm/min^2)
  float millimeters;                 // The remaining distance for this block to be executed in (mm)
  // uint8_t max_override;           // Maximum override value based on axis speed limits

  #ifdef USE_LINE_NUMBERS
    int32_t line_number;
  #endif
} plan_block_t;


// Initialize and reset the motion plan subsystem
void plan_reset();

// Add a new linear movement to the buffer. target[N_AXIS] is the signed, absolute target position
// in millimeters. Feed rate specifies the speed of the motion. If feed rate is inverted, the feed
// rate is taken to mean "frequency" and would complete the operation in 1/feed_rate minutes.
#ifdef USE_LINE_NUMBERS
  void plan_buffer_line(float *target, float feed_rate, uint8_t invert_feed_rate, int32_t line_number);
#else
  void plan_buffer_line(float *target, float feed_rate, uint8_t invert_feed_rate);
#endif

// Called when the current block is no longer needed. Discards the block and makes the memory
// availible for new blocks.
void plan_discard_current_block();

// Gets the current block. Returns NULL if buffer empty
plan_block_t *plan_get_current_block();

// Called periodically by step segment buffer. Mostly used internally by planner.
uint8_t plan_next_block_index(uint8_t block_index);

// Called by step segment buffer when computing executing block velocity profile.
float plan_get_exec_block_exit_speed();

// Reset the planner position vector (in steps)
void plan_sync_position();

// Reinitialize plan with a partially completed block
```

```
void plan_cycle_reinitialize();

// Returns the number of active blocks are in the planner buffer.
uint8_t plan_get_block_buffer_count();

// Returns the status of the block ring buffer. True, if buffer is full.
uint8_t plan_check_full_buffer();

#endif
```

Pemrograman C.12 *Print*

```
/*
  print.h - Functions for formatting output strings
  Part of Grbl

  Copyright (c) 2011-2015 Sungeun K. Jeon
  Copyright (c) 2009-2011 Simen Svale Skogsrud

  Grbl is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  Grbl is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef print_h
#define print_h


void printString(const char *s);

void printPgmString(const char *s);

void printInteger(long n);

void print_uint32_base10(uint32_t n);

// Prints uint8 variable with base and number of desired digits.
void print_unsigned_int8(uint8_t n, uint8_t base, uint8_t digits);

// Prints an uint8 variable in base 2.
void print_uint8_base2(uint8_t n);

// Prints an uint8 variable in base 10.
void print_uint8_base10(uint8_t n);

void printFloat(float n, uint8_t decimal_places);

// Floating value printing handlers for special variables types used in Grbl.
//  - CoordValue: Handles all position or coordinate values in inches or mm reporting.
//  - RateValue: Handles feed rate and current velocity in inches or mm reporting.
```

```
//    - SettingValue: Handles all floating point settings values (always in mm.)
void printFloat_CoordValue(float n);

void printFloat_RateValue(float n);

void printFloat_SettingValue(float n);

// Debug tool to print free memory in bytes at the called point. Not used otherwise.
void printFreeMemory();

#endif
```

Pemrograman C.13 *Probe*

```
/*
   probe.h - code pertaining to probing methods
   Part of Grbl

   Copyright (c) 2014-2015 Sungeun K. Jeon

   Grbl is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.

   Grbl is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.    See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with Grbl.    If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef probe_h
#define probe_h

// Values that define the probing state machine.
#define PROBE_OFF         0 // Probing disabled or not in use. (Must be zero.)
#define PROBE_ACTIVE    1 // Actively watching the input pin.

// Probe pin initialization routine.
void probe_init();

// Called by probe_init() and the mc_probe() routines. Sets up the probe pin invert mask to
// appropriately set the pin logic according to setting for normal-high/normal-low operation
// and the probing cycle modes for toward-workpiece/away-from-workpiece.
void probe_configure_invert_mask(uint8_t is_probe_away);

// Returns probe pin state. Triggered = true. Called by gcode parser and probe state monitor.
uint8_t probe_get_state();

// Monitors probe pin state and records the system position when detected. Called by the
// stepper ISR per ISR tick.
void probe_state_monitor();

#endif
```

Pemrograman C.14 *Protocol*

```
/*
   protocol.h - controls Grbl execution protocol and procedures
   Part of Grbl
```

```
   Copyright (c) 2011-2015 Sungeun K. Jeon
   Copyright (c) 2009-2011 Simen Svale Skogsrud

   Grbl is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.

   Grbl is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef protocol_h
#define protocol_h

// Line buffer size from the serial input stream to be executed.
// NOTE: Not a problem except for extreme cases, but the line buffer size can be too small
// and g-code blocks can get truncated. Officially, the g-code standards support up to 256
// characters. In future versions, this will be increased, when we know how much extra
// memory space we can invest into here or we re-write the g-code parser not to have this
// buffer.
#ifndef LINE_BUFFER_SIZE
  #define LINE_BUFFER_SIZE 80
#endif

// Starts Grbl main loop. It handles all incoming characters from the serial port and executes
// them as they complete. It is also responsible for finishing the initialization procedures.
void protocol_main_loop();

// Checks and executes a realtime command at various stop points in main program
void protocol_execute_realtime();

// Notify the stepper subsystem to start executing the g-code program in buffer.
// void protocol_cycle_start();

// Reinitializes the buffer after a feed hold for a resume.
// void protocol_cycle_reinitialize();

// Initiates a feed hold of the running program
// void protocol_feed_hold();

// Executes the auto cycle feature, if enabled.
void protocol_auto_cycle_start();

// Block until all buffered steps are executed
void protocol_buffer_synchronize();

#endif
```

Pemrograman C.15 *Report*

```
/*
  report.h - reporting and messaging methods
```

```
#ifndef report_h
#define report_h

// Define Grbl status codes.
#define STATUS_OK 0
#define STATUS_EXPECTED_COMMAND_LETTER 1
#define STATUS_BAD_NUMBER_FORMAT 2
#define STATUS_INVALID_STATEMENT 3
#define STATUS_NEGATIVE_VALUE 4
#define STATUS_SETTING_DISABLED 5
#define STATUS_SETTING_STEP_PULSE_MIN 6
#define STATUS_SETTING_READ_FAIL 7
#define STATUS_IDLE_ERROR 8
#define STATUS_ALARM_LOCK 9
#define STATUS_SOFT_LIMIT_ERROR 10
#define STATUS_OVERFLOW 11
#define STATUS_MAX_STEP_RATE_EXCEEDED 12

#define STATUS_GCODE_UNSUPPORTED_COMMAND 20
#define STATUS_GCODE_MODAL_GROUP_VIOLATION 21
#define STATUS_GCODE_UNDEFINED_FEED_RATE 22
#define STATUS_GCODE_COMMAND_VALUE_NOT_INTEGER 23
#define STATUS_GCODE_AXIS_COMMAND_CONFLICT 24
#define STATUS_GCODE_WORD_REPEATED 25
#define STATUS_GCODE_NO_AXIS_WORDS 26
#define STATUS_GCODE_INVALID_LINE_NUMBER 27
#define STATUS_GCODE_VALUE_WORD_MISSING 28
#define STATUS_GCODE_UNSUPPORTED_COORD_SYS 29
#define STATUS_GCODE_G53_INVALID_MOTION_MODE 30
#define STATUS_GCODE_AXIS_WORDS_EXIST 31
#define STATUS_GCODE_NO_AXIS_WORDS_IN_PLANE 32
#define STATUS_GCODE_INVALID_TARGET 33
#define STATUS_GCODE_ARC_RADIUS_ERROR 34
#define STATUS_GCODE_NO_OFFSETS_IN_PLANE 35
#define STATUS_GCODE_UNUSED_WORDS 36
#define STATUS_GCODE_G43_DYNAMIC_AXIS_ERROR 37

// Define Grbl alarm codes.
#define ALARM_HARD_LIMIT_ERROR 1
#define ALARM_SOFT_LIMIT_ERROR 2
#define ALARM_ABORT_CYCLE 3
#define ALARM_PROBE_FAIL 4
#define ALARM_HOMING_FAIL 5
```

```
// Define Grbl feedback message codes.
#define MESSAGE_CRITICAL_EVENT 1
#define MESSAGE_ALARM_LOCK 2
#define MESSAGE_ALARM_UNLOCK 3
#define MESSAGE_ENABLED 4
#define MESSAGE_DISABLED 5
#define MESSAGE_SAFETY_DOOR_AJAR 6
#define MESSAGE_PROGRAM_END 7
#define MESSAGE_RESTORE_DEFAULTS 8

// Prints system status messages.
void report_status_message(uint8_t status_code);

// Prints system alarm messages.
void report_alarm_message(int8_t alarm_code);

// Prints miscellaneous feedback messages.
void report_feedback_message(uint8_t message_code);

// Prints welcome message
void report_init_message();

// Prints Grbl help and current global settings
void report_grbl_help();

// Prints Grbl global settings
void report_grbl_settings();

// Prints an echo of the pre-parsed line received right before execution.
void report_echo_line_received(char *line);

// Prints realtime status report
void report_realtime_status();

// Prints recorded probe position
void report_probe_parameters();

// Prints Grbl NGC parameters (coordinate offsets, probe)
void report_ngc_parameters();

// Prints current g-code parser mode state
void report_gcode_modes();

// Prints startup line
void report_startup_line(uint8_t n, char *line);

// Prints build info and user info
void report_build_info(char *line);

#endif
```

Pemrograman C.16 *Serial*

```
/*
  serial.c - Low level functions for sending and recieving bytes via the serial port
  Part of Grbl

  Copyright (c) 2011-2015 Sungeun K. Jeon
```

```
  Copyright (c) 2009-2011 Simen Svale Skogsrud

  Grbl is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  Grbl is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef serial_h
#define serial_h


#ifndef RX_BUFFER_SIZE
  #define RX_BUFFER_SIZE 128
#endif
#ifndef TX_BUFFER_SIZE
  #define TX_BUFFER_SIZE 64
#endif

#define SERIAL_NO_DATA 0xff

#ifdef ENABLE_XONXOFF
  #define RX_BUFFER_FULL 96 // XOFF high watermark
  #define RX_BUFFER_LOW 64 // XON low watermark
  #define SEND_XOFF 1
  #define SEND_XON 2
  #define XOFF_SENT 3
  #define XON_SENT 4
  #define XOFF_CHAR 0x13
  #define XON_CHAR 0x11
#endif

void serial_init();

// Writes one byte to the TX serial buffer. Called by main program.
void serial_write(uint8_t data);

// Fetches the first byte in the serial read buffer. Called by main program.
uint8_t serial_read();

// Reset and empty data in read buffer. Used by e-stop and reset.
void serial_reset_read_buffer();

// Returns the number of bytes used in the RX serial buffer.
uint8_t serial_get_rx_buffer_count();

// Returns the number of bytes used in the TX serial buffer.
// NOTE: Not used except for debugging and ensuring no TX bottlenecks.
uint8_t serial_get_tx_buffer_count();

#endif
```

Pemrograman C.17 *Settings*

```
/*
  settings.h - eeprom configuration handling
  Part of Grbl

  Copyright (c) 2011-2015 Sungeun K. Jeon
  Copyright (c) 2009-2011 Simen Svale Skogsrud

  Grbl is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  Grbl is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef settings_h
#define settings_h

#include "grbl.h"


// Version of the EEPROM data. Will be used to migrate existing data from older versions of Grbl
// when firmware is upgraded. Always stored in byte 0 of eeprom
#define SETTINGS_VERSION 9  // NOTE: Check settings_reset() when moving to next version.

// Define bit flag masks for the boolean settings in settings.flag.
#define BITFLAG_REPORT_INCHES      bit(0)
// #define BITFLAG_AUTO_START         bit(1) // Obsolete. Don't alter to keep back compatibility.
#define BITFLAG_INVERT_ST_ENABLE   bit(2)
#define BITFLAG_HARD_LIMIT_ENABLE  bit(3)
#define BITFLAG_HOMING_ENABLE      bit(4)
#define BITFLAG_SOFT_LIMIT_ENABLE  bit(5)
#define BITFLAG_INVERT_LIMIT_PINS  bit(6)
#define BITFLAG_INVERT_PROBE_PIN   bit(7)

// Define status reporting boolean enable bit flags in settings.status_report_mask
#define BITFLAG_RT_STATUS_MACHINE_POSITION  bit(0)
#define BITFLAG_RT_STATUS_WORK_POSITION     bit(1)
#define BITFLAG_RT_STATUS_PLANNER_BUFFER    bit(2)
#define BITFLAG_RT_STATUS_SERIAL_RX         bit(3)
#define BITFLAG_RT_STATUS_LIMIT_PINS        bit(4)

// Define settings restore bitflags.
#define SETTINGS_RESTORE_ALL 0xFF // All bitflags
#define SETTINGS_RESTORE_DEFAULTS bit(0)
#define SETTINGS_RESTORE_PARAMETERS bit(1)
#define SETTINGS_RESTORE_STARTUP_LINES bit(2)
#define SETTINGS_RESTORE_BUILD_INFO bit(3)

// Define EEPROM memory address location values for Grbl settings and parameters
// NOTE: The Atmega328p has 1KB EEPROM. The upper half is reserved for parameters and
```

```
// the startup script. The lower half contains the global settings and space for future
// developments.
#define EEPROM_ADDR_GLOBAL         1U
#define EEPROM_ADDR_PARAMETERS     512U
#define EEPROM_ADDR_STARTUP_BLOCK  768U
#define EEPROM_ADDR_BUILD_INFO     942U

// Define EEPROM address indexing for coordinate parameters
#define N_COORDINATE_SYSTEM 6  // Number of supported work coordinate systems (from index 1)
#define SETTING_INDEX_NCOORD N_COORDINATE_SYSTEM+1 // Total number of system stored (from index 0)
// NOTE: Work coordinate indices are (0=G54, 1=G55, ... , 6=G59)
#define SETTING_INDEX_G28    N_COORDINATE_SYSTEM    // Home position 1
#define SETTING_INDEX_G30    N_COORDINATE_SYSTEM+1  // Home position 2
// #define SETTING_INDEX_G92    N_COORDINATE_SYSTEM+2  // Coordinate offset (G92.2,G92.3 not supported)

// Define Grbl axis settings numbering scheme. Starts at START_VAL, every INCREMENT, over N_SETTINGS.
#define AXIS_N_SETTINGS          4
#define AXIS_SETTINGS_START_VAL  100 // NOTE: Reserving settings values >= 100 for axis settings. Up to 255.
#define AXIS_SETTINGS_INCREMENT  10  // Must be greater than the number of axis settings

// Global persistent settings (Stored from byte EEPROM_ADDR_GLOBAL onwards)
typedef struct {
  // Axis settings
  float steps_per_mm[N_AXIS];
  float max_rate[N_AXIS];
  float acceleration[N_AXIS];
  float max_travel[N_AXIS];

  // Remaining Grbl settings
  uint8_t pulse_microseconds;
  uint8_t step_invert_mask;
  uint8_t dir_invert_mask;
  uint8_t stepper_idle_lock_time; // If max value 255, steppers do not disable.
  uint8_t status_report_mask; // Mask to indicate desired report data.
  float junction_deviation;
  float arc_tolerance;

  uint8_t flags;  // Contains default boolean settings

  uint8_t homing_dir_mask;
  float homing_feed_rate;
  float homing_seek_rate;
  uint16_t homing_debounce_delay;
  float homing_pulloff;
} settings_t;
extern settings_t settings;

// Initialize the configuration subsystem (load settings from EEPROM)
void settings_init();

// Helper function to clear and restore EEPROM defaults
void settings_restore(uint8_t restore_flag);

// A helper method to set new settings from command line
uint8_t settings_store_global_setting(uint8_t parameter, float value);
```

```
// Stores the protocol line variable as a startup line in EEPROM
void settings_store_startup_line(uint8_t n, char *line);

// Reads an EEPROM startup line to the protocol line variable
uint8_t settings_read_startup_line(uint8_t n, char *line);

// Stores build info user-defined string
void settings_store_build_info(char *line);

// Reads build info user-defined string
uint8_t settings_read_build_info(char *line);

// Writes selected coordinate data to EEPROM
void settings_write_coord_data(uint8_t coord_select, float *coord_data);

// Reads selected coordinate data from EEPROM
uint8_t settings_read_coord_data(uint8_t coord_select, float *coord_data);

// Returns the step pin mask according to Grbl's internal axis numbering
uint8_t get_step_pin_mask(uint8_t i);

// Returns the direction pin mask according to Grbl's internal axis numbering
uint8_t get_direction_pin_mask(uint8_t i);

// Returns the limit pin mask according to Grbl's internal axis numbering
uint8_t get_limit_pin_mask(uint8_t i);


#endif
```

Pemrograman C. 18 *Spindle Control*

```
/*
  spindle_control.h - spindle control methods
  Part of Grbl

  Copyright (c) 2012-2015 Sungeun K. Jeon
  Copyright (c) 2009-2011 Simen Svale Skogsrud

  Grbl is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  Grbl is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef spindle_control_h
#define spindle_control_h


// Initializes spindle pins and hardware PWM, if enabled.
```

```
void spindle_init();

// Sets spindle direction and spindle rpm via PWM, if enabled.
void spindle_run(uint8_t direction, float rpm);

void spindle_set_state(uint8_t state, float rpm);

// Kills spindle.
void spindle_stop();

#endif
```

Pemrograman C.19 *Stepper*

```
/*
   stepper.h - stepper motor driver: executes motion plans of planner.c using the stepper motors
   Part of Grbl

   Copyright (c) 2011-2015 Sungeun K. Jeon
   Copyright (c) 2009-2011 Simen Svale Skogsrud

   Grbl is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.

   Grbl is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef stepper_h
#define stepper_h

#ifndef SEGMENT_BUFFER_SIZE
  #define SEGMENT_BUFFER_SIZE 6
#endif

// Initialize and setup the stepper motor subsystem
void stepper_init();

// Enable steppers, but cycle does not start unless called by motion control or realtime command.
void st_wake_up();

// Immediately disables steppers
void st_go_idle();

// Generate the step and direction port invert masks.
void st_generate_step_dir_invert_masks();

// Reset the stepper subsystem variables
void st_reset();

// Reloads step segment buffer. Called continuously by realtime execution system.
void st_prep_buffer();
```

```
// Called by planner_recalculate() when the executing block is updated by the new plan.
void st_update_plan_block_parameters();

// Called by realtime status reporting if realtime rate reporting is enabled in config.h.
#ifdef REPORT_REALTIME_RATE
float st_get_realtime_rate();
#endif

#endif
```

Pemrograman C.20 *System*

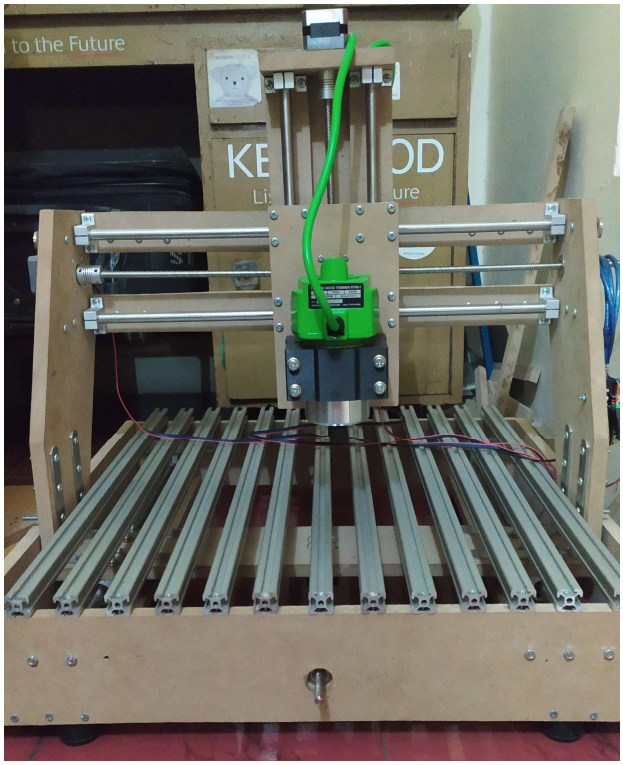
```
/*
  system.h - Header for system level commands and real-time processes
  Part of Grbl

  Copyright (c) 2014-2015 Sungeun K. Jeon

  Grbl is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  Grbl is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with Grbl.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef system_h
#define system_h

#include "grbl.h"

// Define system executor bit map. Used internally by realtime protocol as realtime command flags,
// which notifies the main program to execute the specified realtime command asynchronously.
// NOTE: The system executor uses an unsigned 8-bit volatile variable (8 flag limit.) The default
// flags are always false, so the realtime protocol only needs to check for a non-zero value to
// know when there is a realtime command to execute.
#define EXEC_STATUS_REPORT  bit(0) // bitmask 00000001
#define EXEC_CYCLE_START    bit(1) // bitmask 00000010
#define EXEC_CYCLE_STOP     bit(2) // bitmask 00000100
#define EXEC_FEED_HOLD      bit(3) // bitmask 00001000
#define EXEC_RESET          bit(4) // bitmask 00010000
#define EXEC_SAFETY_DOOR    bit(5) // bitmask 00100000
#define EXEC_MOTION_CANCEL  bit(6) // bitmask 01000000

// Alarm executor bit map.
// NOTE: EXEC_CRITICAL_EVENT is an optional flag that must be set with an alarm flag. When
enabled,
// this halts Grbl into an infinite loop until the user aknowledges the problem and issues a soft-
// reset command. For example, a hard limit event needs this type of halt and aknowledgement.
#define EXEC_CRITICAL_EVENT     bit(0) // bitmask 00000001 (SPECIAL FLAG. See NOTE:)
#define EXEC_ALARM_HARD_LIMIT   bit(1) // bitmask 00000010
#define EXEC_ALARM_SOFT_LIMIT   bit(2) // bitmask 00000100
```

```
#define EXEC_ALARM_ABORT_CYCLE    bit(3) // bitmask 00001000
#define EXEC_ALARM_PROBE_FAIL     bit(4) // bitmask 00010000
#define EXEC_ALARM_HOMING_FAIL    bit(5) // bitmask 00100000

// Define system state bit map. The state variable primarily tracks the individual functions
// of Grbl to manage each without overlapping. It is also used as a messaging flag for
// critical events.
#define STATE_IDLE                0          // Must be zero. No flags.
#define STATE_ALARM               bit(0) // In alarm state. Locks out all g-code processes. Allows settings access.
#define STATE_CHECK_MODE          bit(1) // G-code check mode. Locks out planner and motion only.
#define STATE_HOMING              bit(2) // Performing homing cycle
#define STATE_CYCLE               bit(3) // Cycle is running or motions are being executed.
#define STATE_HOLD                bit(4) // Active feed hold
#define STATE_SAFETY_DOOR         bit(5) // Safety door is ajar. Feed holds and de-energizes system.
#define STATE_MOTION_CANCEL bit(6) // Motion cancel by feed hold and return to idle.

// Define system suspend states.
#define SUSPEND_DISABLE           0          // Must be zero.
#define SUSPEND_ENABLE_HOLD       bit(0) // Enabled. Indicates the cycle is active and currently undergoing a hold.
#define SUSPEND_ENABLE_READY      bit(1) // Ready to resume with a cycle start command.
#define SUSPEND_ENERGIZE          bit(2) // Re-energizes output before resume.
#define SUSPEND_MOTION_CANCEL bit(3) // Cancels resume motion. Used by probing routine.


// Define global system variables
typedef struct {
  uint8_t abort;                  // System abort flag. Forces exit back to main loop for reset.
  uint8_t state;                  // Tracks the current state of Grbl.
  uint8_t suspend;                // System suspend bitflag variable that manages holds, cancels, and safety door.
  uint8_t soft_limit;             // Tracks soft limit errors for the state machine. (boolean)

  int32_t position[N_AXIS];       // Real-time machine (aka home) position vector in steps.
                                  // NOTE: This may need to be a volatile variable, if problems arise.

  int32_t probe_position[N_AXIS]; // Last probe position in machine coordinates and steps.
  uint8_t probe_succeeded;        // Tracks if last probing cycle was successful.
  uint8_t homing_axis_lock;       // Locks axes when limits engage. Used as an axis motion mask in the stepper ISR.
} system_t;
extern system_t sys;

volatile uint8_t sys_probe_state;    // Probing state value.  Used to coordinate the probing cycle with stepper ISR.
volatile uint8_t sys_rt_exec_state;  // Global realtime executor bitflag variable for state management. See EXEC bitmasks.
volatile uint8_t sys_rt_exec_alarm;  // Global realtime executor bitflag variable for setting various alarms.


// Initialize the serial protocol
void system_init();

// Returns if safety door is open or closed, based on pin state.
uint8_t system_check_safety_door_ajar();
```

```
// Executes an internal system command, defined as a string starting with a '$'
uint8_t system_execute_line(char *line);

// Execute the startup script lines stored in EEPROM upon initialization
void system_execute_startup(char *line);

// Returns machine position of axis 'idx'. Must be sent a 'step' array.
float system_convert_axis_steps_to_mpos(int32_t *steps, uint8_t idx);

// Updates a machine 'position' array based on the 'step' array sent.
void system_convert_array_steps_to_mpos(float *position, int32_t *steps);

// CoreXY calculation only. Returns x or y-axis "steps" based on CoreXY motor steps.
#ifdef COREXY
   int32_t system_convert_corexy_to_x_axis_steps(int32_t *steps);
   int32_t system_convert_corexy_to_y_axis_steps(int32_t *steps);
#endif

#endif
```

D. Lampiran Gambar

Gambar D.1 Alat dan bahan



Gambar D.2 Tampak depan mesin CNC Router mini

Gambar D.3 Mesin CNC bersama penulis