Description:

The user manual has information regarding how to operate the machine. This document is for developers by explaining how the code functions and is structured.

There are a lot of iterations of firmware that are applicable to the microcontroller in CP800. Much of the code is shared between all iteration of the firmware. This document will break the firmware by its function and explain each part of it. There is a log of changes written inside the beginning of the main code so developers track when a feature/function was implemented. Of course, that is not the most comprehensive log, so there might be changes that happened between the recorded date. For the sake of documentation, we assume all data recorded on the log is the date of a committed version of the firmware and all changes to between the commits happen on the latest commit.

Main Headers

At the beginning of the code there is a section before the log. This section contains #define for the machine. “model” and “version” are not used functionally as of 8th June 2022. However, PCBver is used to define the pinout for the Dpad and user interface inputs. Since there are two functional PCB being used by CP800 as of 8th June 2022. OUTsense and INsense, is used to defined whether the tank level sensor and flow out sensor are normally open (NO), or normally closed (NC) signal. In the future, more metadata could be defined here so other supporting software could identify the machine and hardware associated with that specific model and version of machine.

#define DEBUG serial print via Arduino COM. This allows a global disable on all serial print functions when we don’t want to be debugging. Note that there are still some serial.print function not controlled by this declaration.

There is one function declaration before the main. This function is used to edit existing processing in the ROM. This function required the following dependency:

The data type of ROMProgram struct

And Arduino basic library.

NOTE: it is planned to modularize the ROM processing code to have a separate file and header.

The section below the comment:

is for the position of each button on the Dpad to the R-2R DAC.

Then the libraries are included below the line:

Four libraries are active as of 8th June 2022

CP800.h and classes.h are defined within the Arduino script folder. LiquidCrystal , OneWire.h + DallasTemperature.h could be installed and updated with libraries manager on Arduino IDE.

Following the comment:

Are the pin assignments for most GPIO for joystick shield, SPI com, motor interface, valves interface, buzzer, fluid sensors.

For the button variables:

UIcommned, is the debounced 8 bits signal for the microcontroller, this includes the buttons digital inputs with the 6 MSBs. The 2nd LSB indicated whether the cursor can travel through the columns, whereas the LSB indicated whether the cursor can travel through the rows.

UIinputs are the raw signal from the 6 inputs from Dpad

De\_button is the debounced signal for UIinputs

Editing is a flag to indicate whether the user is in editing mode or not, see “editing processing” in the user manual.

Menu\_skipupdate is a flag to control whether the menu is being updated or skipped. Might use a better name

These variables are mainly interfacing with the UIcursor and UImap classes which contain the information where the cursor is on the menu, and where in the mapping of the menu. The final inputs are input to the buttonAction subroutine, where the microcontroller reacts to the new inputs every program cycle.

For the LCD, a new library is included, <LiquidCrystal.h>, and a set of pin assignment is #define correspondingly to the object imitation of the lcd, see LiquidCrystal object constructor for how to initiate a lcd object with the library.

The lcd screen used has 4 rows of 20 characters. Four lcdbuffer are made to store each position of the lcd. Four more are made to store the previous screen which are currently not used.

MenuRow is the row number that the cursor/mapping is currently in

Menucolumn is the column number that the cursor/mapping is currently in

Infopage is the page number that the user is currently viewing

MenuTop is a character array with words that are shown on the top menu

Menusetting is a character array with words that are shown on the setting menu

Three varaiables are declared under editing menu. These variables are used when the user is in editing ROM program mode.

EditIndex is the index that indicates the current variable the user is editing. Please see EditsParameters function to see how the entire processing program is flattened and how each data is indexed

EditPara is the value which the index is referring to. There are several types of datatypes, depending on the indexing the data in the array interpreted as different data type, please refer to EditsParameters function for future information

ArrayIndex is the index for the character array, processName[].

EditmenuEnd is a flag for rather the edit menu reached the end or not

Three variables are declared to be used when the user is in the IO check mode in the setting menu, see user manual for how to navigate to the IOcheck menu

OutputCheckAction toggles the action state of each active element of the machine from the menu, typically the component is switched on when this value is 1, such as motor, pump, valve ...etc

IOCheckMode indicates if the program is in the IO check menu. This enables the IOIndex to be changed, because the program has entered IO check mode

IOIndex is the index that shows which input or output the user is currently checking. Refer to inputCheck() and outputCheck() to see how all inputs and outputs are arranged into an array.

ProcessPara is an array of 8 with 20 strings each. String is used here because it is made using the old data format where all parameters are compiled into a String datatype

ProcessTemp is the temperature value for each process

ProcessTime is the duration of each process

Processtotalsteps is the number of steps in each process

ProcessParatSet is the flag that say whether the process has be set (loaded from storage)

ProcessName is the name of the process

Then we will introduce all the data use in processing

For timers:

Long timer is a system timer where it will update every loop

TimeJump is an accumulated varaibles that will increase or decrease the timer values (currently unused)

IniDelay is the delay amount in ms introduce in the for several component when machine is booting up. These include loading processes from ROM, initiating LED, initiating motor.

RefreshTimerA regulate what get update, currently it is at 4 times of refreshintervval which is 2s. These processes are LED, LCD, read temperature

RefreshTImerB regulates what get updated, currently it is at refresh interval rate which is 0.5s. These processes are controlling the motor, controlling the processing, or the cleaning process.

AgitationSpeed determines the speed setting of the motor, this could be expanded to the values of a byte

AgitationTImer is not used

AgitatioTime1 and agitationTime2 are used to create the pattern of motion during processing. The pattern is forward, stop, backward and stop, then repeat.

PrcessState is the flag showing whether the machine is running a processing or not

ProcessNo shows which process from the menu is selected to be processing

Steppingmode shows how the steps in a process are going to progress. Sensor mode means the step progress to the next step depending on the feedback sensor, fixed timer means the step progress to the next only depending on a fixed amount of time.

ProcessStartTime marks the timer instance where a new processing has begun.

There are supposed to be two cleaning processes, they are both currently not programmed. From the order of adding and removing fluid. Clean 1 is having the both add and remove are active depending on a processing array.

Within the processing, each step has timing and parameters that define their actions.

StepState shows whether the processing of the film in this step is done, if done start draining tank fluid and prepare for the next step

StepNo shows sequential order the step is in the processing

StepName shows the name for a step

StepIn and StepOut shows which valve port which are used in the step

StepTime is the duration of the step

StepPush enable/disable the step to be pushed/pulled. Push: increase time of processing of this step, pull: reduce time of processing of this step

Flowin is a flag made to force the chemical to flow in by the user at any time during the step which the processing is currently in

Flowout is a flag to force the chemical to flow out of port 6 during the process, this is used for emergency flow out in case the other valves are somehow blocked during the processing. Please refer to the user manual.

Stepout is supposed to record the value of stepOut the processing was, so that an interrupted operation could be done and yet the processing could continue by pushing this variable back into stepOut (currently unused)

StepTImer1 is use to time the adding chemical stage of the step

StepTimer2 is use to time the rolling processing stage of the step

StepTImer3 is use to time the removing chemical stage of the step

StepParaSet shows whether the step parameters are loaded and ready to go

Push is the value of how much time are increase/decrease (push or pull) for all the pushable steps in the current processing

ErrorMessage is a String for displaying the message when an error incurred

Popuptimer is a time variable for displaying a message on LCD, this is updated to the newest time, so the popup message will display for a duration of time after this variable is updated

EEPROM is one of the options to store and recall processing, all parameters are stored as a byte array.

StrByteLen is currently defined as 200 bytes. Detailed structure of the stored array is in the comment section of the CP800.h tab.

AddrShift is supposed to be a random 8 bits value that shifts the ROM address, in order to achieve wear leveling. The array will have a wrapping order from 256 back to 0. However, the wrapping is not yet programmed. Currently not use.

Load all ibrary required to use SD module on Arduino

SD\_err is the value that returns from attempt to initiate the SD module

Program\_no is the number of processing program stored on the SD card; the program indicator is located at the beginning of each processing program. More details for the data structured store on SD card are described in “process\_data” tab

Loadtype determines how processing program is loaded, either from SD card or EEPROM

ValidROMprogram is an array of flags to indicate which programs stored in the EEMPROM have a valid data structure. If there is no legit program header found in each memory slot, the function will change this array of the corresponding index to 0, so loading on those program entries will be skipped.

Currently temperautre is not use in anyway. This is merely a template for future expansion for now.

#define pin 27 to be the onewire bus

Instantiation of the onewire object to be the define above

Instantiation of the dallas temperature sensor to be the one wire bus device

Tempcurrent and temptarget are the current and target temperature of the sensor. Tempcurrent actually doesn’t need initialization.

Now we instantiate all objects needed to operate the machine with the variable we declared previously:

1. LED
2. Buzzer
3. 6ways valve for both inlet and outlet
4. Motor
5. Bathpump, currently no corresponding hardware
6. Digital level sensor for inside the tank and along the outlet tube
7. Analog level sensors are not used
8. The heating element using AC power
9. Joystick shield
10. Dpad interface as the UI
11. The cursor location on the screen is the user
12. The location displayed on the screen is the menu

# CP800.h

This header tab is made to contains classes and global variable for those classes in the CP800 series

Libraries used in the classes : pitches.h, EEPROM.h, LiquidCrystal.h

Pitches.h: has the buzzer value to make different notes and function to generate pre-defined tune

LiquidCrystal is the libaray for display on the LCD

EEPROM is the library for accessing the EEPROM on Arduino

ArrayLength is to calculate the size of an array using sizeof()

Zero\_crossed is the flag to indicate whether the AC input of the heating has crossed zero. This allows a fixed energy cycle for the PWM cycle within a certain time error

Maxcol is the variable that allows users to shift left and right in the menu to see different information

Rowstack is the data matrix that stored the row coordinate of both the user cursor and menu map

Colstack is the data matrix that stored the column coordinate of both the user cursor and menu map

Actionstack is the value for the action that the machine is taking, this is normally on the last page in the menu, where the user has to pick a function or run a program. The previously two stacks are for the location of the menu through each layer of the selection on the menu.

LcdStr is the array of 4 strings to be displayed on the LCD

IspageEnd is the flag for indicating the menu had reached an end page, a page where you have to select an action and no more option to be select after that

Pagerow is the number of rows on the current menu page. Usually 4 rows, given the screen has 4 rows, but main menu and program selection has more, and allow scrolling down

The camera struct contain the coordinate of the camera and the size in each dimension to be seen by the camera. Imagine it to be the view range of a person on the map.

## Class UIcursor

Private variables include:

\_inputsRef is the pointer to the input variable

\_inputs is the variable that take the new inpur

\_previnputs is the variable that stored the previous inputs

\_rowlimit is restricted to row 3 (4th row) , because cursor cannot move out of the screen

The private function limit() restricts the value of row and column between 0 to the upperlimit, so far only row has an upper limit for the cursor

Row, col, layer fully describe the location of the cursor on the map

Instantiate the class by passing a pointer of a byte

Initialize the class by taking the input from the pointer and make previous input the same

Public function to edit the current location of the cursor to any value

The update function is the core function of the class. It updates the position of the cursor by reading and interpreting the buttons inputs, when the update is called.

The button shift defined in header will now be used to isolate the A and B button, these buttons confirm or cancel an action.

When press A:

1. Stored the current row and column position to the stack
2. If this is the end page, row value is the action option, if not, we increment the layer
3. Then reset row and column position to 0 as we are either doing a new action,
4. Since no page is beyond layer 3, restrict layer increment when reached end, this could be redundant if mapping and tracking is done better

When press B:

1. Reset current position to zero in stack
2. Decrement layer
3. Load previous layer position
4. Clear the previous location from stack

When press other buttons

1. Increment or decrement row value if press up and down while row shift bit is enabled (\_inputs&1)
2. Increment or decrement column value if press left and right while row shift bit is enabled (\_inputs>>1&1)
3. Check and limit row and column according to the boundary of the menu page

Always, update input to previous input at the end

## Class UImap

This class is very similar to UIcursor, in fact, it could be position to generalise both UImap and UIcursor class into one class.

All except the inclusion of camera struct is the same as UIcursor. Camera structs allow UImap not only the position but the range of location that is visible

Initialize class with camera values, and camera sizes are fixed

Exact same function as UIcursor

Update mirror the UIcursor

The camera will also be updated.

## Class Buzzer

This class required the pitches.h header

\_in\_pin, \_rh\_pin, \_rl\_pin, are the Dpad input pin, high voltage reference pin, low voltage reference pin respectively

\_init\_volt, \_init\_rh, \_init\_rl, are the initialise reading, in Begin() from the input pin, high voltage reference pin, and low voltage reference pin respectively

\_new\_volt, \_new\_rh, \_new\_rl are the new reading from the input pin, high voltage reference pin, and low voltage reference pin respectively

\_dVolt is a variable to calcualte the difference between two votlage reading

\_old\_volt, is the \_new\_volt in the previous update

\_threshold define the deadband in which the program ignore the fluctuation of the voltage reading in the input pin

\_shift\_ratio is a ratio to represent relative voltage change, the full equation wil be defined in Update()

\_ref\_range is the variable used to store the value between the high and low reference voltage

\_button\_reg\_set is a flag is indicate whether a button is pressed, 0 is pressed, given input are pulled up

\_edgetrigger is the flag for an edge has been triggered

\_edgetiming is the timing when the edge triggered.

NOTE: might be neater to compress these pin reading into a data array, and the update operate could iterate through the variable instead of calling each variable individually.

Buzzer has 4 private variables;  
\_buzz\_pin contain the pin number for the buzzer

\_vol\_pin, controls the MOSFET that boost the current to the buzzer

\_welcomeMelody is a two notes array that make noise when bootup,

\_welcomeMelodyDuration is the timer period for each note in the welcome tune

Two constructors

1. Without volume control
2. With volume control

Play note function only play one note with one duration

Play array function cycle thru all the note in array and match the duration

Welcome subroutine play the welcome tune using play array function

There is a subroutine that play nothing

Initialize class by setup buzz and volume pin as output in Arduino

Vol\_hi function connext the buzzer ground with a low resistor value using MOSFET, low resistor more current thus louder

Vol\_lo function cut off the buzzer ground with a low resistor value using MOSFET

## Class Dpad\_analog

Dpad status\_reg records the status of the Dpad which are the 6 buttons inputs plus row and column shift enable bit at the end

Constructor with inputs to \_in\_pins, two analog pin for the high logics voltage and low logics voltage

Initialisation routine define

1. \_in\_pin as input, output from DAC IC
2. \_rh\_pin as input pulled up, for high voltage
3. \_rl\_pin as input, externally pulled down, for low voltage
4. Read all inputs pin to initialize values

Checkupdate() routine return 1 when new value is \_threshold difference from the old values

Ispressed() return 0 when a button is pressed

# Future development

* Optimization of RAM usage
  + Remove global variables that are not used
  + More modularized code and use local variable.
  + Better way to store tune and menu option (mainly const chr/int)
* More buttons and better structure button classes
  + Currently the respond is are only 6 buttons, directions + AB, this should be expanded to respond to a more complex controller and allow more action from the user
  + Allow a more efficient way to quick travel to another location or return to previous location, A systematic set of coordinates in the mapping might help.
  + The structure for the mapping on the menu and cursor should remain separated, imagine the cursor is the user’s character, and it is navigating the menu as if it is an RPG. Both the POV of the environment and the chr have to react to the user’s inputs.
* Better ROM read/write addressing. Currently the ROM programs are referring to a set of fixed addresses. This could result in an imbalance number of read/write for some memory unit, and thus worn out those units quicker than others.
  + New writing methods should consider the health of each ROM unit and maximize their life by using dynamic indexing, to avoid overwriting existing ROM units.
  + The new method should also repeat old information in a new region. It would be ideal to reuse a step from another processing cycle than write a new step, since many step between different processing can be shared, such as rinse with water.
  + maybe using wear leveling technique to use each ROM cell evenly. However, this might nullify the ROM.update() function where data is only update when there is a change to it
  + Better still, to use a data compressing method that maximised reusing existing variables and trimmed data to be stored.
* Better code organization
  + Tabs can be combined/separated according to their function/hardware/other arrangement
  + There could be a classification of different header for different purpose, to further modalized the codes, this might allow an entire feature to be reuse in another project.
* UI
  + Generise UIcursor and UImap with a navigation class that has position value