COOLANT NOZZLE POSITIONING SYSTEM FOR CNC MACHINING CENTERS

By

ARUN PRAKAASH C

SSN COLLEGE OF ENGINEERING,
KALAVAKKAM, CHENNAI - 603110

Under guidance of,

Mr. MITHUN M
R&D,
ACE MANUFACTURING SYSTEMS Ltd.,
BANGALORE - 560058

At (Hybrid mode)

ACE MANUFACTURING SYSTEMS Ltd.,
BANGALORE - 560058

Duration:

28th March, 2023 – 29th July, 2023

CONTENTS

1. PROJECT TITLE:

Coolant Nozzle Positioning System for CNC Machines.

2. BROAD SUBJECT:

To develop a coolant nozzle positioning system to improve the performance of the existing coolant nozzle system by adding an extra feature, i.e., nozzle direction control by which the impact of coolant on the material is made precise with up to +/- 0.225 degrees scale accuracy.

3. OBJECTIVES OF THE PROPOSED PROJECT:

1. To develop a Mechanical system consisting of stepper motor and coolant nozzle.
2. To develop an Embedded system consisting of STM32 family microcontroller.
3. To interface the mechanical system and embedded system and creation of an open loop control system.

4. DELIVERABLES:

The outcome of this project will be leading to creation of a more efficient, having an advanced control option set coolant nozzle system. The precision of the coolant impact on the target will drastically increase with millimeter scale accuracy.

5. HARDWARE USED:

A. Blue Pill (STM32F103C8T6):

Blue Pill is a 32-bit Arduino compatible development board that features the STM32F103C8T6, a member of the STM32 family of ARM Cortex-M3 core microcontrollers. Blue pill is shown in figure 1 and the pin details is shown in figure 2.

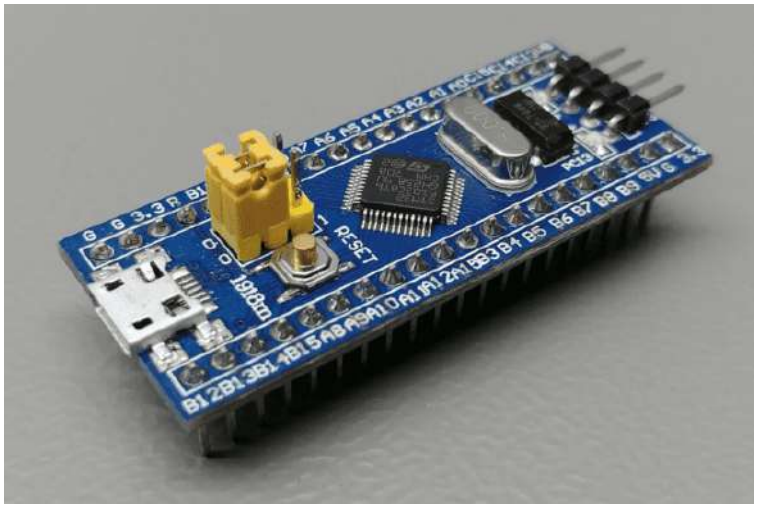

Figure 1

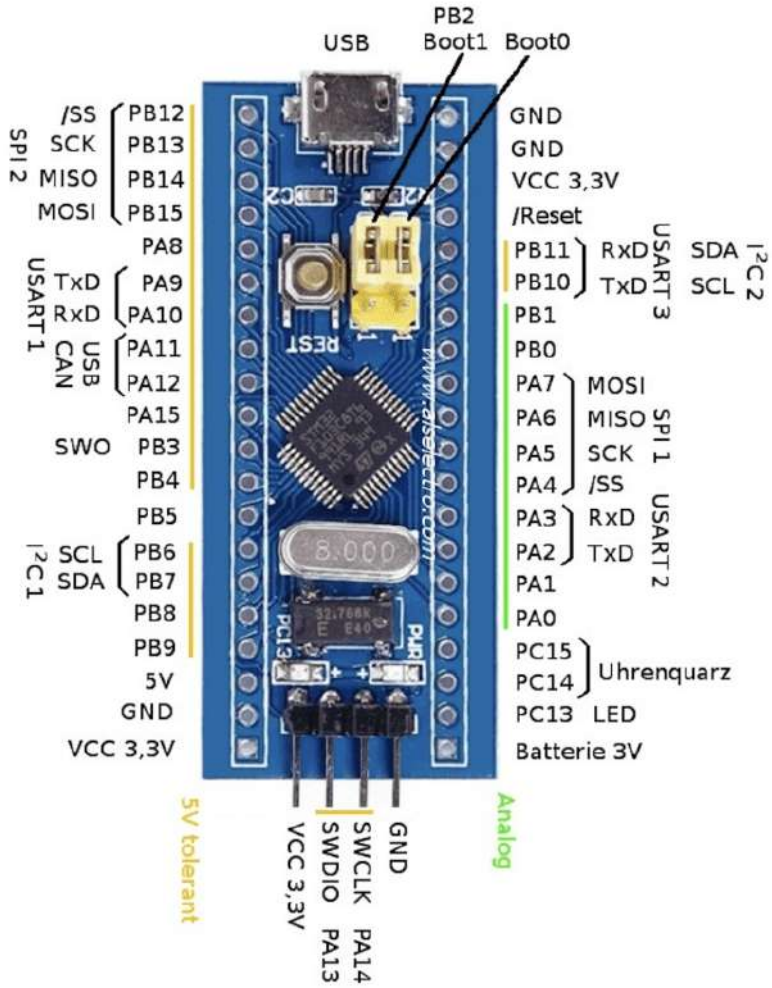

Figure 2

B. NEMA17 Stepper Motor (with worm gearbox):

NEMA 17 stepper motor has a full step angle of 1.8 degrees and is a 200 steps-per-revolution four-phase unipolar permanent-magnet stepper-motor. This gives a minimum torque of 50 N/cm is used to control the direction of the coolant nozzle.
NEMA 17 stepper motor is shown in figure 3.

Figure 3

C. TB6600 Stepper Motor Driver:

TB6600 Stepper motor driver is a cheap and effective device which provides micro stepping ability to a stepper motor. It is suitable for driving 2 phase and 4 phase hybrid stepper motors. The motor driver compatible with any microcontroller providing a 5V signal.

Features and Specifications of TB6600 Stepper Motor Driver Module:
This section mentions some of the applications of TB6600 stepper motor driver.

1. Operating voltage: 9-40V DC.
2. Output Current: 0.7-4.0A.
3. Pulse input frequency up to 20kHz.
4. 5V levels input signal.
5. 200-6400 pulse per revolution.
6. Logic signal current: 8-15mA.
7. Output current selectable in 8-steps via DIP switches.
8. Suitable for 2 and 4 phase motors.
9. Over current and over heat protection.
10. Inputs are optically isolated.

TB6600 stepper motor driver is shown in figure 5 and 6.
In this project the TB6600 mode is set to 8 micro step mode and set to current supply of 1.5A (max-1.7A).

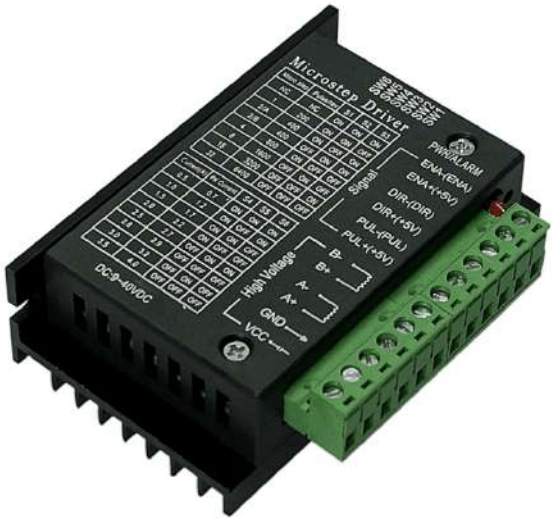

Figure 4

Figure 5

6. DESIGN OVERVIEW:

In figure 6, the required angle calculation methodology is shown.

In figure 7, the main signal flowchart through the hardware is depicted.

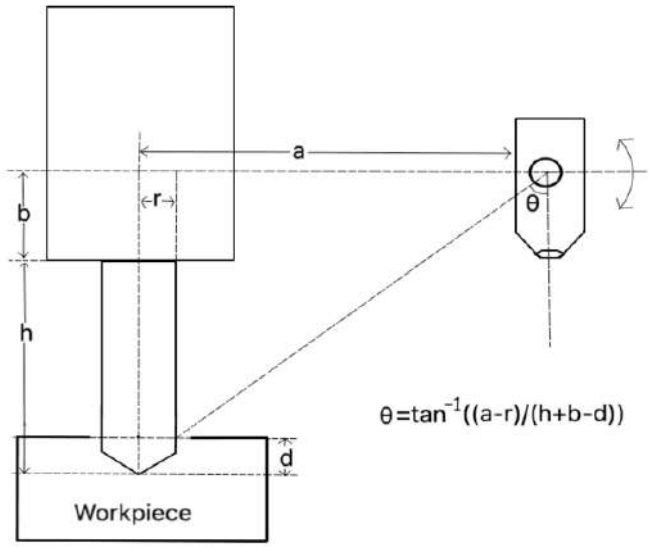

Figure 6

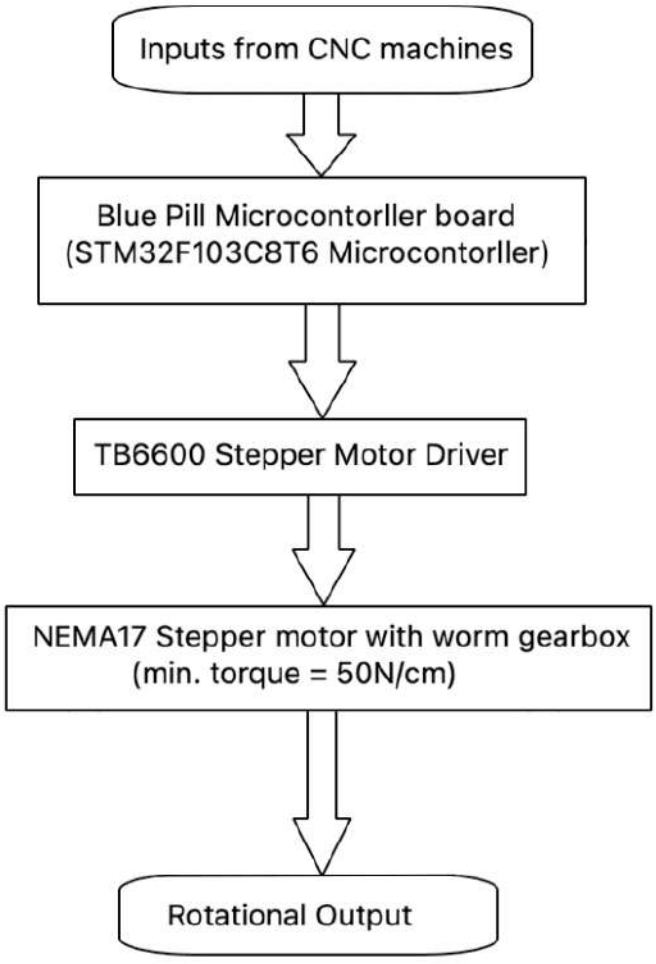

Figure 7

7. CIRCUIT DIAGRAM:

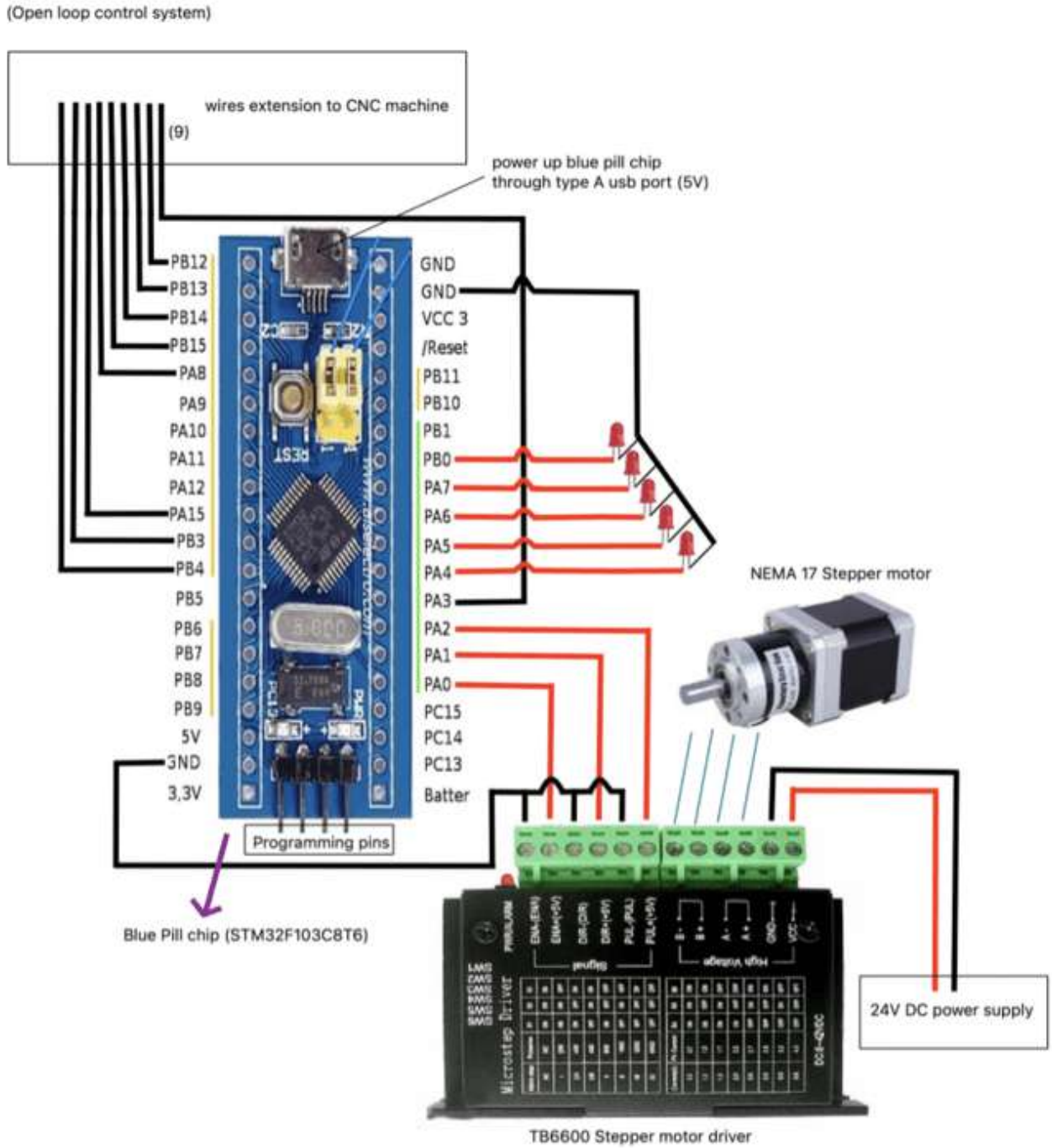

Figure 8

8. SOFTWARE USED:

STM32CubeIDE software is used to program the blue pill chip (STM32F103C8T6 microprocessor).

STM32CubeIDE is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors. It is based on the Eclipse®/CDT™ framework and GCC toolchain for the development, and GDB for the debugging.

SOURCE CODE:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "math.h"
#define enable GPIO_PIN_0
#define direction GPIO_PIN_1
#define pulse GPIO_PIN_2
#define power GPIO_PIN_4
#define mode GPIO_PIN_3
#define B0 GPIO_PIN_15
#define B1 GPIO_PIN_8
#define B2 GPIO_PIN_15
#define B3 GPIO_PIN_14
#define B4 GPIO_PIN_13
#define B5 GPIO_PIN_12
#define input6_enable GPIO_PIN_3

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
```

```
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */
  HAL_GPIO_WritePin(GPIOA, direction, GPIO_PIN_RESET);
  int step_previous = 0;

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
 if (HAL_GPIO_ReadPin(GPIOB, power) == 0)
 {
 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET); //////////////////////
 if (HAL_GPIO_ReadPin(GPIOB, mode) == 0) // advanced mode
 {
```

```
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET); ////////////////////////
int b0;
int b1;
int b2;
int b3;
int b4;
int b5;
if (HAL_GPIO_ReadPin(GPIOA, B0) == 0) //0
{
b0 = 1;
}
else
{
b0 = 0;
}
if (HAL_GPIO_ReadPin(GPIOA, B1) == 0) //1
{
b1 = 1;
}
else
{
b1 = 0;
}
if (HAL_GPIO_ReadPin(GPIOB, B2) == 0) //2
{
b2 = 1;
}
else
{
b2 = 0;
}
if (HAL_GPIO_ReadPin(GPIOB, B3) == 0) //3
{
b3 = 1;
}
else
{
b3 = 0;
}
if (HAL_GPIO_ReadPin(GPIOB, B4) == 0) //4
{
b4 = 1;
}
else
{
b4 = 0;
}
if (HAL_GPIO_ReadPin(GPIOB, B5) == 0) //5
{
b5 = 1;
}
else
{
b5 = 0;
}
if ( HAL_GPIO_ReadPin(GPIOA, input6_enable) == 0)
{
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET); ////////////////////////
int step_current = 0;
int step = 0;
```

```
int conversion = 0;
int binary[] = {b0, b1, b2, b3, b4, b5};
for (int j = 0; j<6; j++)
{
conversion += binary[j]*pow(2,j);
}
step_current = conversion*8;
if (step_current > step_previous)
{
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET); //////////////////////
HAL_GPIO_WritePin(GPIOA, direction, GPIO_PIN_SET);
HAL_Delay(1);
step = step_current - step_previous;
for (int a = 0; a < step; a++)
{
HAL_GPIO_WritePin(GPIOA, pulse, GPIO_PIN_SET);
HAL_Delay(0.005);
HAL_GPIO_WritePin(GPIOA, pulse, GPIO_PIN_RESET);
HAL_Delay(0.005);
}
}
else if (step_current < step_previous)
{
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET); //////////////////////
HAL_GPIO_WritePin(GPIOA, direction, GPIO_PIN_RESET);
HAL_Delay(1);
step = step_previous - step_current;
for (int a = 0; a < step; a++)
{
HAL_GPIO_WritePin(GPIOA, pulse, GPIO_PIN_SET);
HAL_Delay(0.005);
HAL_GPIO_WritePin(GPIOA, pulse, GPIO_PIN_RESET);
HAL_Delay(0.005);
}
}
step_previous = step_current;
}
}
else // oscillating mode
{
if (step_previous != 0)
   {
HAL_GPIO_WritePin(GPIOA, direction, GPIO_PIN_RESET);
HAL_Delay(1);
int step_last = step_previous;
for(int a = 0; a<step_last; a++)
{
   HAL_GPIO_WritePin(GPIOA, pulse, GPIO_PIN_SET);
HAL_Delay(0.005);
HAL_GPIO_WritePin(GPIOA, pulse, GPIO_PIN_RESET);
HAL_Delay(0.005);
}
step_previous = 0;
   }
else
{
HAL_GPIO_WritePin(GPIOA, direction, GPIO_PIN_SET);
HAL_Delay(1);
for( int i = 0; i<400; i++)
{
```

```
 HAL_GPIO_WritePin(GPIOA, pulse, GPIO_PIN_SET);
 HAL_Delay(0.005);
 HAL_GPIO_WritePin(GPIOA, pulse, GPIO_PIN_RESET);
 HAL_Delay(0.005);
 }
 HAL_GPIO_WritePin(GPIOA, direction, GPIO_PIN_RESET);
 HAL_Delay(1);
 for( int i = 0; i<400; i++)
 {
 HAL_GPIO_WritePin(GPIOA, pulse, GPIO_PIN_SET);
 HAL_Delay(0.005);
 HAL_GPIO_WritePin(GPIOA, pulse, GPIO_PIN_RESET);
 HAL_Delay(0.005);
 }
 }
 }
 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET); ////////////////////////
 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET); ////////////////////////
 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET); ////////////////////////
 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET); ////////////////////////
 //step_previous = step_current;
 }
 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET); ////////////////////////
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
```

```
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_4
                        |GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);

  /*Configure GPIO pins : PA0 PA1 PA2 PA4
                         PA5 PA6 PA7 */
  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_4
                        |GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : PA3 PA8 PA15 */
  GPIO_InitStruct.Pin = GPIO_PIN_3|GPIO_PIN_8|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PB0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pins : PB12 PB13 PB14 PB15
                         PB3 PB4 */
  GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15
                        |GPIO_PIN_3|GPIO_PIN_4;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
```

```
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

The above mentioned source code is used to program the STM32F103C8T6 microcontroller. Using this, a binary file (.bin) and a hexadecimal file (.hex) can be generated. This binary file is flashed into the microcontroller using ST-link V2 debugger and programmer.

9. WORKING AND FUNCTIONALITY:

As per the circuit diagram mentioned in this document the system has been made and the blue pill chip was programmed using the previously mentioned source code.

The system is designed such that the processing power required is distributed for both the CNC machine controller and the blue pill chip. Also designed such that it can operate upon two modes, they are Oscillating mode and Advanced mode.

- **Oscillating mode:**

When this mode is set the coolant nozzle oscillates in required angle range. In the current developed prototype the range is 0-90 degrees.

- **Advanced mode:**

When this mode is set, according to the changes in the tool available in the spindle the required angle is set accordingly. The parameters considered here are the tool's height and radius.

CNC MACHINE SETUP:

The system is designed such that the output, that we get from CNC machine will be 9 serial line ON/OFF signals, which is used for different functions as described below,

1 signal line for Mode-1 signal
1 signal line for Mode-2 signal
1 signal line for Mode-2 data input enable signal
6 signal lines for data transfer from CNC machine controller to the blue pill chip. These 6 bits data inputs are the encoded CNC machine computed values.

The following equations are predefined in the CNC machine controller,
a = *constant, set once - changes according to the CNC machine type*
b = *constant, set once - changes according to the CNC machine type*
x = tan-inverse((a-r)/(h+b))
y = x/1.8
z = roundoff(y)
where r is the radius of the tool and h is the height of the tool

Two tables are created and predefined in the CNC machine controller. These tables can be considered as the database from where we can take the required tool parameter values which is then used for processing the previously mentioned equations, and finally a 6 bit data required as output from CNC machine controller is generated.

This 6 bit parallel data is the encoded signal that will be transferred to the blue pill chip.

Model tables that are created in the CNC machine controller:

Table 1:

Tool number	Radius of the tool	Height of the tool	x	y	z

Table 2:

z	Binary equivalent data (6-bits)
1	000001
2	000010
3	000011
4	000100
5	000101
6	000110
7	000111
8	001000
9	001001
10	001010
11	001011
12	001100
13	001101
14	001110
15	001111
16	010000
17	010001
18	010010
19	010011
20	010100
21	010101
22	010110
23	010111
24	011000
25	011001
26	011010
27	011011
28	011100
29	011101
30	011110
31	011111
32	100000
33	100001
34	100010
35	100011
36	100100
37	100101
38	100110
39	100111
40	101000
41	101001
42	101010
43	101011
44	101100
45	101101
46	101110
47	101111
48	110000
49	110001
50	110010

Once the x, y and z values are found out using the predefined equations that are available in the CNC machine controller, the binary equivalent data (6-bits) for 'z' is found out by mapping to table-2 data and this 6-bit data will be given as output from CNC machines which will be an input data for blue pill chip.

Table 2 is predefined and need not to be changed unless if we want to change the range from 0-90 degrees to some other required range. Table 1 contents changes according to the tools that are used in the CNC machine.

Table 1 - Example:

Tool number	Radius of the tool	Height of the tool	x	y	z
1	20	60	58.57	32.538	33
2	40	120	43.26	24.033	24
3	60	180	31.32	17.400	17
4	20	10	71.56	39.755	40
5	80	280	19.98	11.100	11

CNC Machine & Developed Embedded Hardware Interfacing:

CNC machine produces 9 line signal outputs, which are given as input to the developed embedded system. According to the inputs the embedded system gets, the signals are converted into getting a final rotational mechanical output at stepper motor.

Inputs to the developed embedded system are:

1 signal line for Mode-1 signal
1 signal line for Mode-2 signal
1 signal line for Mode-2 data input enable signal
6 signal lines for data transfer from CNC machine controller to the blue pill chip. These 6 bits data inputs are the encoded CNC machine computed values.

Note: In this project the TB6600 stepper motor driver mode is set to 8 micro step mode and is set to output current supply of 1.5A (max-1.7A). Thus, we will get 0.225 degrees as minimum step angle from NEMA 17 stepper motor.

Prototype Images:

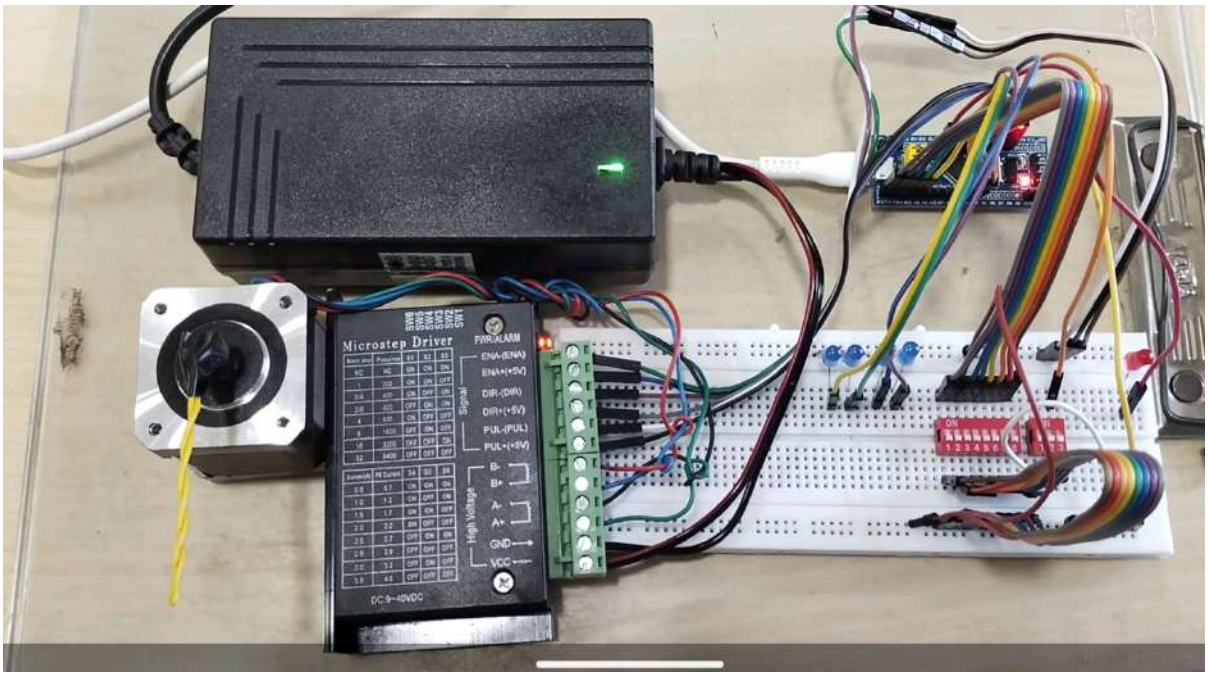

Figure 9

Figure 10

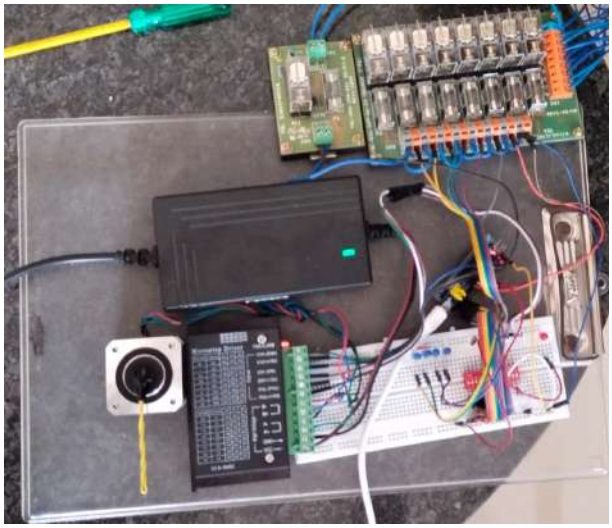

Figure 11

Figure 12

10. CONCLUSION:

A coolant nozzle positioning system is developed. The current prototype model can be further enhanced and has potential to be modified in future as per custom requirements.