ДОДАТОК А

Графічна частина кваліфікаційної роботи

Магістерська робота 2021

Харківський національний університет радіоелектроніки

Кафедра АПОТ

Кваліфікаційна робота магістра

Моделі часових керуючих автоматів з гнучкою логікою на основі мікроконтролерів

Магістрантки групи СКСм-20-1
Кислянської Дар"ї
Вікторівни

Керівник:
доц. каф. АПОТ
Шкіль О.С

Харків 2021

Kharkov National University of Radioelectronics

Магістерська робота 2021

Постановка задачі

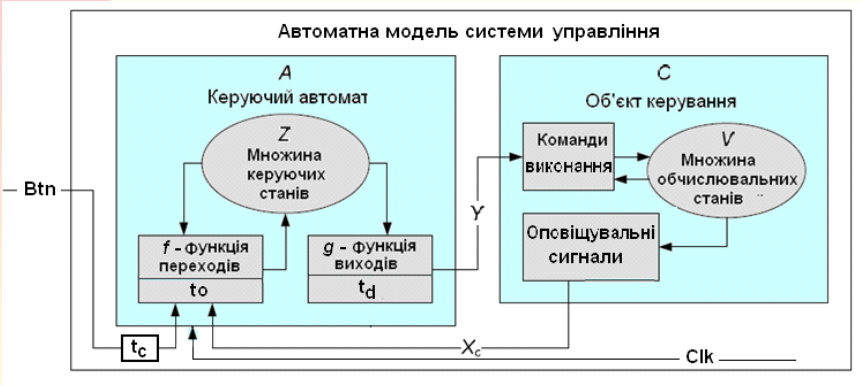
- Метою роботи є побудова моделі часового керуючого автомата з гнучкою логікою та реалізація цієї моделі в мікроконтролерних пристроях.

  Для досягнення поставленої мети необхідно вирішити такі задачі:
- провести аналіз методів проектування систем логічного управління;
- провести аналіз методів побудови керуючих мікропрограмних автоматів з гнучкою логікою;
- розробити формат мікрокоманди часового автомата з гнучкою логікою;
- виконати програмну реалізацію та моделювання керуючого автомата з гнучкою логікою для мікроконтролера у запропонованій моделі;
- провести порівняння розробленої моделі мікроконтролерного автомата з гнучкою логікою з традиційною моделлю часового автомата на мові програмування CI.

Kharkov National University of Radioelectronics 2

Автоматна модель системи управління

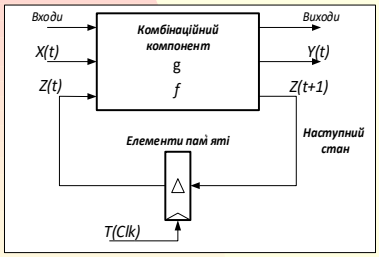
- Z – множина керуючих станів; Btn – множина зовнішніх подій;
- X_C – множина оповіщувальних сигналів; Clk – синхропослідовність



- t_c – часові (вхідні) обмеження; to - таймаути; t_d - вихідні затримки.

Часовий структурний автомат

- Для реалізації системи управління реального часу використовується модель часового автомата (timed FSM), яка дозволяє враховувати вплив метричного часу на переходи між технічними станами системи управління за рахунок введення узагальненої часової змінної T.
- Часовий автомат $W = (X, Y, Z, f, g, z_0, T)$



Функція виходів $Y(t) = g(X(t), Z(t))$

Функція переходів $Z(t+1) = f(X(t), Z(t))$

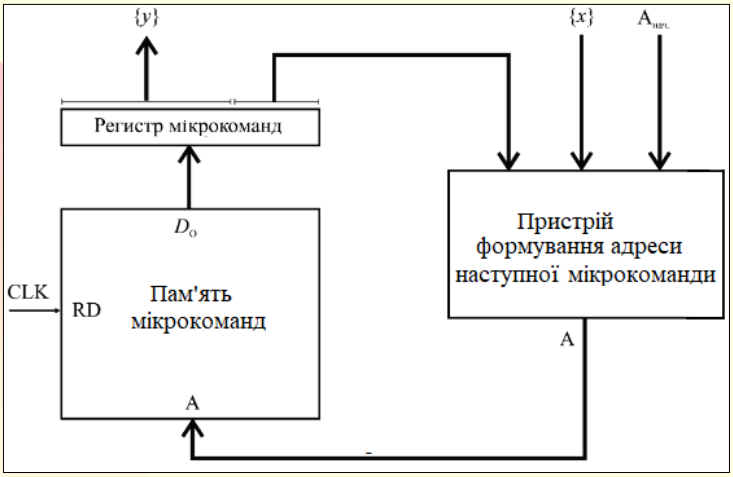
Новий стан $Z(t+1) \equiv Z(t)$

Часовий структурний автомат $Z(t+1) = f(X(t), Z(t), T)$

$Y(t) = g(X(t), Z(t), T)$

- Часова змінна T складається з трьох параметрів: часові обмеження t_c (timing constraints), вхідні таймаути t_{to} (timeouts) і вихідні затримки t_d (output delays), які іноді називаються вихідними таймаутами.

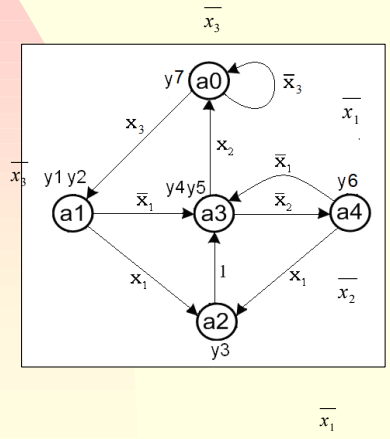
Мікропрограмний автомат з гнучкою логікою



- X – вхідні сигнали; Y - вихідні сигнали; A – адреса мікрокоманди.

Графова та таблична модель автомата Мура

- **Граф переходів** і пряма **структурна таблиця** автомата Мура

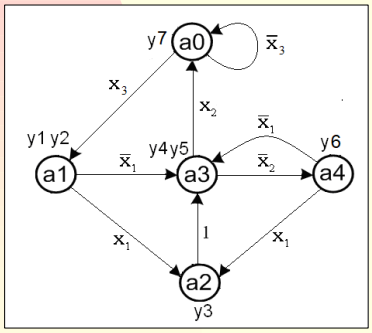


№	a_m	y_m	x_{ms}	a_s
0	a_0	y_7	x_3	a_0 a_1
1	a_1	y_1y_2	x_1	a_2 a_3
2	a_2	y_3	1	a_3
3	a_3	y_4y_5	x_2	a_0 a_4
4	a_4	y_6	x_1	a_2 a_3

Формат мікрокоманди автомата з гнучкою логікою

- Структура мікрокоманди автомата з гнучкою логікою

№	y_{m1}	y_{m2}	…	y_{mk}	x_{m1}	x_{m2}	…	x_{mn}	a_{m1}	a_{m2}	…	a_{ms}

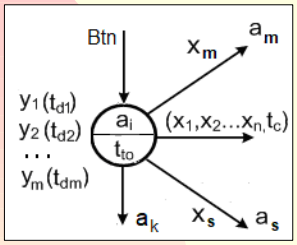


№	a_m	y_{m1}	y_{m2}	x_m	$a_{x=0}$	$a_{x=1}$
0	a_0	y_7	-	x_3	0	1
1	a_1	y_1	y_2	x_1	3	2
2	a_2	y_3	-	x_3	3	3
3	a_3	y_4	y_5	x_2	4	0
4	a_4	y_6	-	x_1	3	2

№	y_{m1}	y_{m2}	x_m	$a_{x=0}$	$a_{x=1}$
0	111	000	11	000	001
1	001	010	01	011	010
2	011	000	11	011	011
3	100	101	10	100	000
4	110	000	01	011	010

Модель часового автомата Мура з гнучкою логікою

- Графова і таблична модель часового автомата Мура



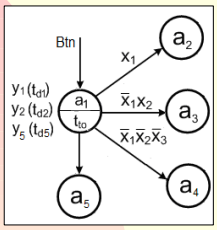
a_i		Y_i		X_n		Event	
a_i	to_i	y_m	t_{dm}	x_n	a_n	Btn	a_k

- Формат мікрокоманди часового автомата з гнучкою логікою

a_i				Y_i		X_n		Event	
adr	to_i	N_y	N_x	y_m	t_{dm}	x_n	adr(a_n)	Btn	adr(a_k)

Кодування мікрокоманди часового автомата Мура з гнучкою логікою

- Графова і таблична модель часового автомата Мура



to1 - 5 sec; Ny=3;
t_{d1} – 2 sec; t_{d2} – 1 sec; t_{d5} – 0 sec;
01 – xi; 10 – not(xi);
00 – xi не використовується.
Присутність події – 01,
відсутність події – 00.

- Формат закодованої мікрокоманди часового автомата з гнучкою логікою

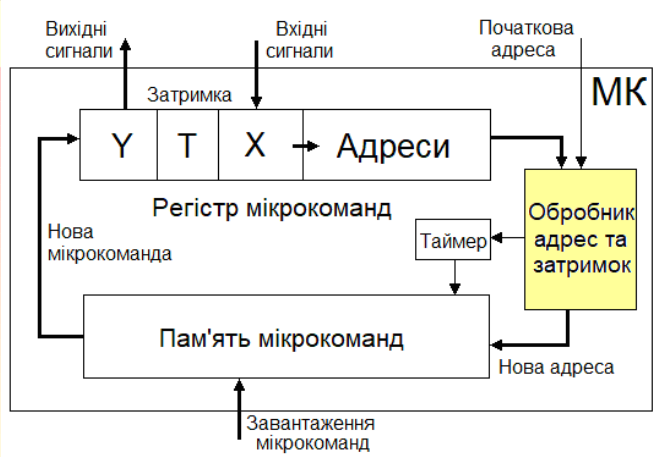
a_i				y_1		y_2		y_3	
a_1	5	3	3	y_1	2	y_2	1	y_5	0

$X(a_{12})$				$X(a_{13})$				$X(a_{14})$				Event	
x_1	-	-	a_2	$\sim x_1$	x_2	-	a_3	$\sim x_1$	$\sim x_2$	$\sim x_3$	a_4	Btn	a_5

0001	101	011	011	001	010	010	001	101	000

01	00	00	0010	10	01	00	0011	10	10	10	0100	01	0101

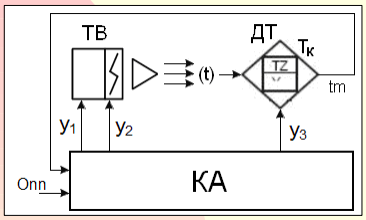
Структура мікроконтролерного автомата з гнучкою логікою



- X – вхідні сигнали; Y – вихідні сигнали; A – адреса мікрокоманди;
- T – час знаходження у відповідному стан (затримка).

Система автоматичного управління тепловентилятором

- Об'єктом проектування є система управління тепловентилятором.



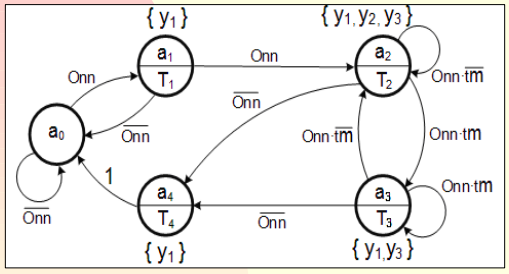
Onn = {0, 1} – вхідний сигнал включення живлення тепловентилятора;
St = {0, 1} – вхідний сигнал включення режиму нагріву;
y1 = {0, 1} – вихідний сигнал включення вентилятора;
y2 = {0, 1} – вихідний сигнал включення нагрівального елементу;
y3 = {0, 1} – вихідний сигнал опитування датчика температури;
tm = {0, 1} – вихідний сигнал датчика температури,

Стани керуючого автомату

Стан a0 (тепловентилятор виключений), Onn=0, y1 = y2 = y3 = 0.
Стан a1 (режим включення та розгону вентилятора), Onn=1, tm=0. y1=1, затримка T1.
Стан a2 (режим нагріву та подачі теплого повітря), Onn=1, y1 = y2 = y3 = 1, затримка опитування T2.
Стан a3 (режим охолодження нагрівального елементу), Onn=1, y1 = y3 = 1, y2 =0, затримка T3 .
Стан a4 (режим повного охолодження нагрівального елементу), Onn=0, y2 = 1, y1 = y3 = 0, затримка T4 .

Графова та таблична модель керуючого автомата Мура в управлінні тепловентилятором

- Граф переходів та пряма структурна таблиця автомата Мура.



№	a_m	T_m	y_m	x_{ms}	a_s
0	a_0	-	-	Onn	a_0 a_1
1	a_1	$T_1 \cdot \overline{tm}$	y_1	Onn	a_2 a_0
2	a_2	T_2	y_1 y_2 y_3	Onn Onn · tm	a_2 a_3 a_4
3	a_3	$T_3 \cdot \overline{tm}$	y_1 y_3	Onn Onn · tm	a_2 a_3 a_4
4	a_4	T_4	y_1	1	a_0

T1=3 сек – проміжок часу для розгону вентилятора до необхідної швидкості;
T2=1 сек – проміжок часу між опитуваннями датчика температури при нагріві;
T3=5 сек – проміжок часу між опитуваннями датчика температури при охолодженні;
T4=15 сек – проміжок часу для повного охолодження нагрівального елементу.

Кодована пряма структурна таблиця та мікрокоманди автомата з гнучкою логікою

- Закодована пряма структурна таблиця автомата Мура →

№	a_m	T_m	y_m	x_{ms}	a_s
0	a_0 000	-	-	(10 00) Onn (01 00)	a_0 (000) a_1 (001)
1	a_1 001	T_1 0011	y_1 (01) $\overline{Onn}$	Onn (01 00) (10 00)	a_2 (010) a_0 (000)
2	a_2 010	T_2 0001	y_1 (01) y_2 (10) y_3 (11)	Onn (01 10) Onn · tm (01 01) (10 00)	a_2 (010) a_3 (011) a_4 (100)
3	a_3 011	T_3 0101	y_1 (01) y_3 (11) $\overline{Onn}$	Onn (01 10) Onn · tm (01 01) (10 00)	a_2 (010) a_3 (011) a_4 (100)
4	a_4 100	T_4 1111	y_1 (01)	1 (00 00)	a_0 (000)

a_1				Onn tm | a_n			
000	0000	000	010	10 00	000	01 00	001

a_1		y_1	Onn tm | a_n			
001	0011	001	010	01 00	01 00	010	10 00	000

a_2		y_1	y_2	y_3	Onn tm | a_n					
010	0001	011	011	01 00	10 00	11 00	01 10	010	01 01	011	01 10	100

a_3		y_1	y_3	onn tm | a_n						
011	0101	010	011	01 00	11 00	01 10	010	01 01	011	01 10	100

a_4		y_1	onn tm | a_n		
100	1111	001	001	01 00	00 00	000

Формат закодованих мікрокоманд часового автомата з гнучкою логікою

Автоматний шаблон, функція переходів

- Програмная реалізація на мові CI для МК STM32

```
/* TRANSITIONS FUNCTION START */
switch(state) {
        case a0:
        //Special part of code that initiate the transition without extrnal buttons and sensors
                // #ifdef DEBUG_MODE    Onn = 1;
                #endif
                if (Onn)   { nextState = a1 }
                else   { nextState = a0; }
        break;

        case a1:
                if(Onn) { nextState = a2; }
                else{ nextState = a0; }
        break;

        case a2:
                //Special code for debug mode
                //helps to count the necessary temp value (50 degrees by Celcius)
          //and get the correct nextState value without external buttons and sensors
                //#ifdef DEBUG_MODE
                  if( debugTemp >= TEMP_TO_COOL ) { tm = 1; }
                  else {debugTemp += TEMP_TO_HEAT/2;  //each 10 degrees
                        tm = 0; }
                  #endif
                if( Onn && tm )   { nextState = a3;  }
                else if ( Onn && !tm )    { nextState = a2;  }
                else{ nextState = a4; }
        break;
```

Автоматний шаблон, функції виходів та нового стану і затримок

- Програмная реалізація на мові CI для МК STM32 F103C8T6

```
/* OUTPUTS FUNCTION START */
switch(state)
{
        case a0:
          Y1_OFF;Y2_OFF;Y3_OFF;
        break;

        case a1:
          Y1_ON;Y2_OFF;Y3_OFF;
        break;

        case a2:
          Y1_ON;Y2_ON;Y3_ON;
        break;

        case a3:
          Y1_ON;Y2_OFF;Y3_ON;
        break;

        case a4:
          Y1_ON;Y2_OFF;Y3_OFF;
        break;
        default:
          Y1_OFF;Y2_OFF;Y3_OFF;
        break;
}
```

```
/* DELAYS FUNCTION START */

state = nextState;      //transit to nextState value

//select the necessary Timeout for current state
switch(state)
{
  case a0: Timeout = T0; break;

  case a1: Timeout = T1;  break;

  case a2: Timeout = T2;  break;

  case a3: Timeout = T3;  break;

  case a4: Timeout = T4;  break;

  default:  Timeout = T0;   break;
}

 #ifdef DEBUG_MODE
    HAL_GPIO_WritePin( GPIOA, GPIO_PIN_3,
              (GPIO_PinState)Onn );
    HAL_GPIO_WritePin( GPIOA, GPIO_PIN_4,
              (GPIO_PinState)tm  );
#endif
```


Модель пам'яті мікрокоманд автомата з гнучкою логікою в МК STM32 F103C8T6

```
typedef struct
{
        uint8_t curState          :3;
        uint8_t curStateDelay     :4;
        uint8_t outputsAmount     :2;
        uint8_t arcAmount         :3;
        uint8_t firstOutputValue  :2;
        uint8_t firstOutputDelay  :2;
        uint8_t secOutputValue    :2;
        uint8_t secOutputDelay    :2;
        uint8_t thirdOutputValue  :2;
        uint8_t thirdOutputDelay  :2;
        uint8_t firstInputValue   :4;
        uint8_t firstNextState    :3;
        uint8_t secInputValue     :4;
        uint8_t secNextState      :3;
        uint8_t thirdInputValue   :4;
        uint8_t thirdNextState    :3;
}curStateMicroCommand[STATES_COUNT
];
```

```
pstatic volatile curStateMicroCommand STATES =
{
        { 0, 1, 0, 2,   3, 3, 3, 3, 3, 3,   8, 0, 4, 1, 15, 7 },
        { 1, 3, 1, 2,   1, 0, 3, 3, 3, 3,   4, 2, 8, 0, 15 ,7  },
        { 2, 1, 3, 3,   1, 0 ,2, 0, 3, 0,   6, 2, 5, 3, 8, 4 },
        { 3, 5, 2, 3,   1, 0, 3, 3, 3, 0,   6, 2, 5, 3, 8, 4  },
        { 4,15,1, 1,   1, 0, 3, 3, 3, 3,   0, 0, 15, 7, 15, 7 }
};

static volatile uint8_t state;        //current state
static volatile uint8_t nextState;    //next state
static volatile uint16_t Timeout;     //timeout in seconds
```

Програмний код обробника адрес та затримок для автомата з гнучкою логікою

- Програмная реалізація на мові CI для МК STM32 F103C8T6

```
/* INPUT STATEMENTS WITH TRANSITION */
//All the InputValue fields should be processed!
if( INPUT_MISSING_NUMBER != STATES[state].
   firstInputValue || DEFAULT_TRANSITION ==
   STATES[state].firstInputValue )
    { nextState = STATES[state].firstNextState; }
elseif( INPUT_MISSING_NUMBER !=
STATES[state].
      secInputValue || DEFAULT_TRANSITION ==
         STATES[state].secInputValue )
    { nextState = TATES[state].secNextState;}
else if( INPUT_MISSING_NUMBER !=
STATES[state].
    thirdInputValue || DEFAULT_TRANSITION ==
         STATES[state].thirdInputValue)
   { nextState = STATES[state].thirdNextState; }
else //default state
        { nextState = a4; }
/* END INPUT STATEMENTS WITH TRANSITION
*/
```

```
/* GPIO OUTPUT */

//Checking and write the output value to the GPIO
//All the OutputDelay fields should be processed!
//
if( OUTPUT_MISSING_NUMBER != STATES[state].firstOutputDelay )
  {HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);}
else
 {HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET);}

if( OUTPUT_MISSING_NUMBER != STATES[state].secOutputDelay )
 {HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6,GPIO_PIN_SET);}
else
 {HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6,GPIO_PIN_RESET);}

if( OUTPUT_MISSING_NUMBER != STATES[state].thirdOutputDelay)
 {HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7,GPIO_PIN_SET);}
else
{ HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7,GPIO_PIN_RESET);}

/* END GPIO OUTPUT */
```

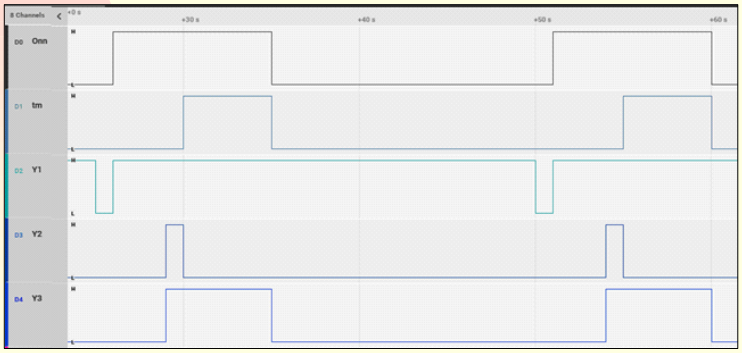
Логічні аналізатори

- Для отримання даних з системи, що розробляється, буде використано логічний аналізатор Saleae Logic 8. Параметрів даного аналізатора логіки більше ніж достатньо для отримання сигналів з каналів або передачі даних протоколів. А його ціна у 10 USD дозволяє визнати його найдешевшим способом аналізу трафіку та даних у будь-яких проектах з невеликою частотою. Даний логічний аналізатор підтримується ПЗ, що безкоштовно надається компанією Saleae.

Результати моделювання керуючого автомата з жорсткою логікою

- Для демонстрації працездатності мікроконтролерної моделі часового керуючого автомату Мура з жорсткою логікою виконаємо обхід всіх вершин графової моделі з урахуванням часових параметрів за циклом :

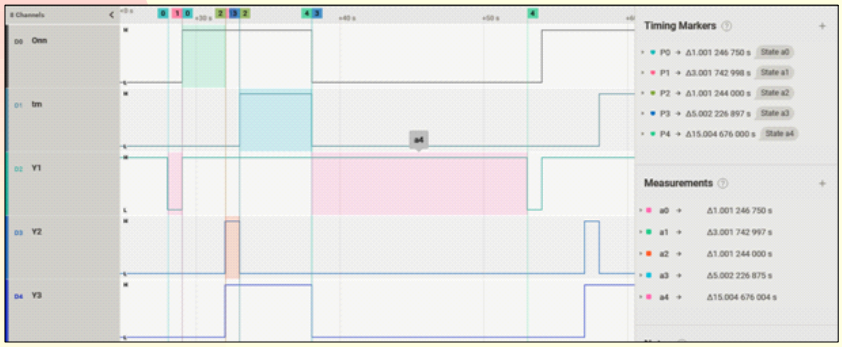
a0 – **a1**(y1, 3 сек.) – **a2**(y1, y2, y3, 1 сек.) – **a3**(y1, y3, 5 сек.) – **a2**(y1, 3 сек.) – **a4**(y1, 15 сек.) – **a0** .



Результати моделювання керуючого автомата з гнучкою логікою

- Для демонстрації працездатності мікроконтролерної моделі часового керуючого автомату Мура гнучкою логікою виконаємо обхід всіх вершин графової моделі за таким же циклом :

a0 – **a1**(y1, 3 сек.) – **a2**(y1, y2, y3, 1 сек.) – **a3**(y1, y3, 5 сек.) – **a2**(y1, 3 сек.) – **a4**(y1, 15 сек.) – **a0** .



Результати порівняння моделей автоматів з жорсткою та гнучкою логікой

- Результати часового моделювання в цілому співпали з результатами поведінкового моделювання

Спосіб реалізації	Кількість рядків коду	Об'єм файлу прошивки	Кількість рядків файлу прошивки
Класичний автоматний шаблон	133	18 кБайт (17.9)	412
Автомат із гнучкою логікою	71	18 кБайт (17.4)	400

ВИСНОВКИ

- В ході виконання роботи проведений аналіз методів проектування систем логічного управління та аналіз методів побудови керуючих мікропрограмних автоматів з гнучкою логікою.
- Розроблений формат мікрокоманди часового керуючого автомату Мура з гнучкою логікою та структура реалізації керуючого автомату Мура з гнучкою логікою в мікроконтролері.
- Моделі часових автоматів представлені на мові програмування CI для мікроконтролера STM32F103 у формі автоматного шаблону та у формі набору мікрокоманд з примусовою адресацією для гнучкого автомата. Виконано поведінкове моделювання запропонованих моделей, результати представлені у формі часових діаграм за допомогою програмного логічного аналізатора.
- Проведений порівняльний аналіз розробленої моделі мікроконтролерного автомата з гнучкою логікою з традиційною моделлю часового автомата у формі автоматного шаблону на мові програмування CI.

ДОДАТОК Б

Програмний код автоматного шаблону

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.h
  * @brief          : Header for main.c file.
  *                   This file contains the common defines of the application.
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2019 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */

/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */

/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/
/* USER CODE BEGIN Private defines */
      //States
```

```
      //
      #define a0  0
      #define a1  1
      #define a2  2
      #define a3  3
      #define a4  4

      //Timeouts in ms
      //
      #define      T0  1000     //Timeout for a0
      #define T1  3000    //Timeout for a1
      #define T2  1000    //Timeout for a2
      #define T3  5000    //Timeout for a3
      #define T4  15000   //Timeout for a4

      //Outputs
      //
      #define Y1_ON       HAL_GPIO_WritePin( GPIOA, GPIO_PIN_5, GPIO_PIN_SET );
      //Define for y1 output OFF function on GPIO
      #define Y2_ON       HAL_GPIO_WritePin( GPIOA, GPIO_PIN_6, GPIO_PIN_SET );
      //Define for y2 output OFF function on GPIO
      #define Y3_ON       HAL_GPIO_WritePin( GPIOA, GPIO_PIN_7, GPIO_PIN_SET );
      //Define for y3 output OFF function on GPIO
      #define Y1_OFF      HAL_GPIO_WritePin( GPIOA, GPIO_PIN_5, GPIO_PIN_RESET );
//Define for y1 output ON function on GPIO
      #define Y2_OFF      HAL_GPIO_WritePin( GPIOA, GPIO_PIN_6, GPIO_PIN_RESET );
//Define for y2 output ON function on GPIO
      #define Y3_OFF      HAL_GPIO_WritePin( GPIOA, GPIO_PIN_7, GPIO_PIN_RESET );
//Define for y3 output ON function on GPIO
/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
/*_____________________________________________________________________________*/

/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2019 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "tim.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

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

```
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define DEBUG_MODE //define for testing mode (without transitions with external
buttons and sensors)
/* USER CODE END PD */

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

/* USER CODE END PM */

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

/* USER CODE BEGIN PV */
static volatile uint8_t state;
static volatile uint8_t nextState;

static volatile uint16_t Timeout; //volatile identifier for using the variable value
in timer interrupt

#ifdef DEBUG_MODE
	static volatile uint8_t Onn; //variable for transition launching without
external buttons and sensors
	static volatile uint8_t tm; //variable for transition to a3 state without
external buttons and sensors
#endif

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_NVIC_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 */
	const uint8_t TEMP_TO_HEAT  = 20; //const value for cool temperature sensor
  const uint8_t TEMP_TO_COOL  = 50; //const value for heat temperature sensor

	#ifdef DEBUG_MODE
		uint8_t debugTemp = TEMP_TO_HEAT; //variable for getting the temperature
without external sensor
	#endif
  /* 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();
  MX_TIM3_Init();

  /* Initialize interrupts */
  MX_NVIC_Init();                                                    //Allow all the
necessary interrupts
  /* USER CODE BEGIN 2 */
       HAL_TIM_Base_Start_IT(&htim3); //Allow to start the transition function

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
              /* TRANSITIONS FUNCTION START */
              switch(state)
              {
                     case a0:

                     //Special part of code that initiate the transition without extrnal buttons and sensors
                            //
                            #ifdef DEBUG_MODE
                                   Onn = 1;
                            #endif

                            if(Onn)
                            {
                                   nextState = a1;
                            }
                            else
                            {
                                   nextState = a0;
                            }

                     break;

                     case a1:

                            if(Onn)
                            {
                                   nextState = a2;
                            }
                            else
                            {
                                   nextState = a0;
                            }

                     break;

                     case a2:

                            //Special code for debug mode
                            //helps to count the necessary temp value (50 degrees by Celcius)
                       //and get the correct nextState value without external buttons and sensors
                            //
                            #ifdef DEBUG_MODE
                         if( debugTemp >= TEMP_TO_COOL )
                             {
                                    tm = 1;
                             }
                             else
                             {
                                    debugTemp += TEMP_TO_HEAT/2; //each 10 degrees
                                    tm = 0;
                             }
                            #endif

                            if( Onn && tm )
                            {
                                   nextState = a3;
```

```
                }
                else if ( Onn && !tm )
                {
                        nextState = a2;
                }
                else
                {
                        nextState = a4;
                }
        break;

        case a3:

                //Test code for direct transition from state a3 to state a4
                //
                #ifdef DEBUG_MODE
                        Onn = 0;
                        tm  = 0;
            debugTemp = TEMP_TO_HEAT;
                #endif

                if( Onn && tm )
          {
                        nextState = a3;
                }
                else if ( Onn && !tm )
                {
                        nextState = a2;
                }
                else
                {
                        nextState = a4;
                }
        break;

        case a4:
                nextState = a0;
        break;

        default:
                nextState = a0;
        break;
}
/* TRANSITIONS FUNCTION END */

/* OUTPUTS FUNCTION START */
switch(state)
{
        case a0:
                Y1_OFF;Y2_OFF;Y3_OFF;
        break;

        case a1:
                Y1_ON;Y2_OFF;Y3_OFF;
        break;

        case a2:
                Y1_ON;Y2_ON;Y3_ON;
        break;

        case a3:
                Y1_ON;Y2_OFF;Y3_ON;
        break;

        case a4:
                Y1_ON;Y2_OFF;Y3_OFF;
        break;
        default:
                Y1_OFF;Y2_OFF;Y3_OFF;
        break;
}
```

```
            /* OUTPUTS FUNCTION END */

    /* USER CODE END WHILE */
  }
    /* USER CODE BEGIN 3 */

  /* USER CODE END 3 */
}

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

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
  /** 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.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 192;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  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_DIV2;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    Error_Handler();
  }
}
/**
  * @brief NVIC Configuration.
  * @retval None
  */
static void MX_NVIC_Init(void)
{
  /* TIM3_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(TIM3_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(TIM3_IRQn);
}

/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) //Timer interrupt handler(callback)
{
       if(htim->Instance == TIM3)
       {
              HAL_TIM_Base_Stop_IT(&htim3); //Stop the timer and forbid the interrupt (for insurance)

              /* DELAYS FUNCTION START */
              state = nextState; //transit to nextState value
```

```
		//select the necessary Timeout for current state
		//
		switch(state)
		{
			case a0:
				Timeout = T0;
			break;

			case a1:
				Timeout = T1;
			break;

			case a2:
				Timeout = T2;
			break;

			case a3:
				Timeout = T3;
			break;

			case a4:
				Timeout = T4;
			break;

			default:
				Timeout = T0;
			break;
		}

		//Set the Onn and tm signals as the outputs and show them on waveform for test aims
		//
		#ifdef DEBUG_MODE
			HAL_GPIO_WritePin( GPIOA, GPIO_PIN_3, (GPIO_PinState)Onn );
			HAL_GPIO_WritePin( GPIOA, GPIO_PIN_4, (GPIO_PinState)tm  );
		#endif

		/* DELAYS FUNCTION END */

		__HAL_TIM_SET_AUTORELOAD(&htim3, Timeout); //Reload the timer

		HAL_TIM_Base_Start_IT(&htim3); //return the interrupt mode and resume the main code part
	}
}
/* 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 */
  /* 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,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

ДОДАТОК В

Програмний код автомата з гнучкою логікою

```
/* USER CODE BEGIN Header */
/**
******************************************************************************
*
  * @file           : main.h
  * @brief          : Header for main.c file.
  *                   This file contains the common defines of the application.

******************************************************************************
*
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2019 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
******************************************************************************
  */
/* USER CODE END Header */

/* Define to prevent recursive inclusion
-------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------*/
#include "stm32f4xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
```

```
/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */
/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */
/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */
/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */
/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/

/* USER CODE BEGIN Private defines */

#define STATES_COUNT                     5
#define     DEFAULT_TRANSITION           0
#define ONE_SEC                         1000 //Timeout to seconds definition

#define OUTPUT_MISSING_NUMBER            3 //Special define for output that
might not be shown on GPIO
#define INPUT_MISSING_NUMBER            15 //Special define for input statements that
might not be cheked on states

/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */

/********************* (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2019 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause
license,
  * the "License"; You may not use this file except in compliance with
the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *

******************************************************************************
  */
/* USER CODE END Header */
/* Includes
------------------------------------------------------------------*/
#include "main.h"
#include "tim.h"
#include "gpio.h"

/* Private includes
----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

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

/*
 * bitfield struct as the COMMAND PARSER
 * each field has the command bitcode sequence
*/
typedef struct
{
     uint8_t curState                :3;
  uint8_t curStateDelay         :4;
  uint8_t outputsAmount               :2;
     uint8_t arcAmount                          :3;
     uint8_t firstOutputValue   :2;
     uint8_t firstOutputDelay   :2;
     uint8_t secOutputValue          :2;
     uint8_t secOutputDelay          :2;
     uint8_t thirdOutputValue   :2;
```

```
      uint8_t thirdOutputDelay   :2;
      uint8_t firstInputValue   :4;
      uint8_t firstNextState          :3;
      uint8_t secInputValue      :4;
      uint8_t secNextState            :3;
      uint8_t thirdInputValue         :4;
      uint8_t thirdNextState          :3;
}curStateMicroCommand[STATES_COUNT];

/* USER CODE END PTD */

/* Private define
------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define DEBUG_MODE //Special define for unit test execution without
sensors

#ifdef DEBUG_MODE //Special state defines for unit test executions
without sensors
      #define a0 0
      #define a1 1
      #define a2 2
      #define a3 3
      #define a4 4
#endif
/* USER CODE END PD */

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

/* USER CODE END PM */

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

/* USER CODE BEGIN PV */

/*
 * Memory mapping example
 * Each bitfield (a structure with fields that have concrete bit
amount)
 * decribes the microcommand section that will be processed by the
command parser
 * (microcommand execution engine)
 * The memory mapping is unified structure with possibility not to
operate with
 * microcommand sections that have the maximum field value
 * e.g. if the firstOutputDelay (2 bits) field has the value of 3
 * the engine will not set the output value to "1" on GPIO
*/
static volatile curStateMicroCommand STATES =
{
      { 0, 1,0,2, 3,3,3,3,3,3, 8,0,4,1,15,7  },
      { 1, 3,1,2, 1,0,3,3,3,3, 4,2,8,0,15,7  },
      { 2, 1,3,3, 1,0,2,0,3,0, 6,2,5,3,8,4   },
      { 3, 5,2,3, 1,0,3,3,3,0, 6,2,5,3,8,4   },
      { 4,15,1,1, 1,0,3,3,3,3, 0,0,15,7,15,7 }
```

```
};

static volatile uint8_t state;              //current state
static volatile uint8_t nextState;  //next state
static volatile uint16_t Timeout;           //timeout in seconds

#ifdef DEBUG_MODE
     static volatile uint8_t OnnAndTm; //Special variable of input
signals for unit test executions without external sensors
#endif
/* USER CODE END PV */

/* Private function prototypes
-----------------------------------------------*/
void SystemClock_Config(void);
static void MX_NVIC_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 */
     const uint8_t TEMP_TO_HEAT  = 20; //The default temperature value
as the constant
  const uint8_t TEMP_TO_COOL  = 50; //The constant temperature value
of heatered fan

     #ifdef DEBUG_MODE
          uint8_t debugTemp = TEMP_TO_HEAT; //Special variable to
count the temperature without the sensor
     #endif
  /* 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();
  MX_TIM3_Init();

  /* Initialize interrupts */
  MX_NVIC_Init();
  /* USER CODE BEGIN 2 */
     HAL_TIM_Base_Start_IT(&htim3); //Allow to start the transition
function

  /* USER CODE END 2 */

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

          /* CODE FOR UNIT TEST WITHOUT SENSORS */
          #ifdef DEBUG_MODE
               if( a0 == STATES[state].curState )
               {
               //Initiate the transition to the next state
               // Onn = 01; tm = 00 so the OnnAndTm will be 0100 (4)
                    //
                    OnnAndTm = 4;
               }

               if( a2 == STATES[state].curState )
               {
     //Unit test to check the heating and initiate the transition from a2 to a3
                    //
                    if( debugTemp >= TEMP_TO_COOL )
                    {
                         OnnAndTm = 5;
                    }
                    else
                    {
                         OnnAndTm   = 6;
                         debugTemp += TEMP_TO_HEAT/2;
                    }
               }

               if( a3 == STATES[state].curState )
               {
          //Inititate the transition from a3 to a4 with set the variables to the default value
          //OnnAndTm = 1000 (nextState should be a0); debugtemp = 20;
                    //
                    OnnAndTm  = 8;
                    debugTemp = TEMP_TO_HEAT;
               }
          #endif
          /* END CODE FOR UNIT TEST WITHOUT SENSORS*/
```

```
            /* INPUT STATEMENTS WITH TRANSITION */

            //Checking whether the OnnAndTm should be processed to get
the nextStateValue
            //All the InputValue fields should be processed!
            //
            if( INPUT_MISSING_NUMBER != STATES[state].firstInputValue ||
DEFAULT_TRANSITION == STATES[state].firstInputValue )
            {
                  nextState = STATES[state].firstNextState;
            }

            else if( INPUT_MISSING_NUMBER != STATES[state].secInputValue
|| DEFAULT_TRANSITION == STATES[state].secInputValue )
            {
                  nextState = STATES[state].secNextState;
            }

            else if( INPUT_MISSING_NUMBER !=
STATES[state].thirdInputValue || DEFAULT_TRANSITION ==
STATES[state].thirdInputValue)
            {
                  nextState = STATES[state].thirdNextState;
            }

            else //default state
            {
                  nextState = a4;
            }
            /* END INPUT STATEMENTS WITH TRANSITION */

            /* GPIO OUTPUT */

            //Checking and write the output value to the GPIO
            //All the OutputDelay fields should be processed!
            //
            if( OUTPUT_MISSING_NUMBER != STATES[state].firstOutputDelay)
            {
                  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);
            }
            else
            {
                  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET);
            }

            if( OUTPUT_MISSING_NUMBER != STATES[state].secOutputDelay )
            {
                  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6,GPIO_PIN_SET);
            }
            else
            {
                  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6,GPIO_PIN_RESET);
            }

            if( OUTPUT_MISSING_NUMBER != STATES[state].thirdOutputDelay)
            {
                  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7,GPIO_PIN_SET);
            }
```

```
            else
            {
                  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7,GPIO_PIN_RESET);
            }
            /* END GPIO OUTPUT */

            /* END COMMAND PARSER */

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

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

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
  /** 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.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 192;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  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_DIV2;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) !=
HAL_OK)
  {
```

```
    Error_Handler();
  }
}

/**
  * @brief NVIC Configuration.
  * @retval None
  */
static void MX_NVIC_Init(void)
{
  /* TIM3_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(TIM3_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(TIM3_IRQn);
}

/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) //Timer
interrupt handler(callback)
{
      if(htim->Instance == TIM3)
      {
            HAL_TIM_Base_Stop_IT(&htim3); //Stop the timer and forbid
the interrupt (for insurance)

            state = nextState; //Initiate the transition
            Timeout = STATES[state].curStateDelay * ONE_SEC; //Get the
state timeout and convert it to the ms

            /*
             * Special part of code to show on GPIO the OnnAndTm values
             * The third bit of variable shows the Onn signal result (0
or 1)
             * The first bit of variable shows the tm signal result (0
or 1)
            */
            #ifdef DEBUG_MODE
                  //Write the Onn value with parsing the third bit and
tm value with parsing the first bit
                  //
                  HAL_GPIO_WritePin( GPIOA, GPIO_PIN_3, (GPIO_PinState)
((OnnAndTm&4)>>2) );
                  HAL_GPIO_WritePin( GPIOA, GPIO_PIN_4, (GPIO_PinState)
(OnnAndTm&1) );
            #endif

             __HAL_TIM_SET_AUTORELOAD(&htim3, Timeout); //Reload the
timer for interrupt restarting

            HAL_TIM_Base_Start_IT(&htim3); //return the interrupt mode
      }
}
/* 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 */

  /* 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,
     tex: printf("Wrong parameters value: file %s on line %d\r\n",
file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/**************(C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```