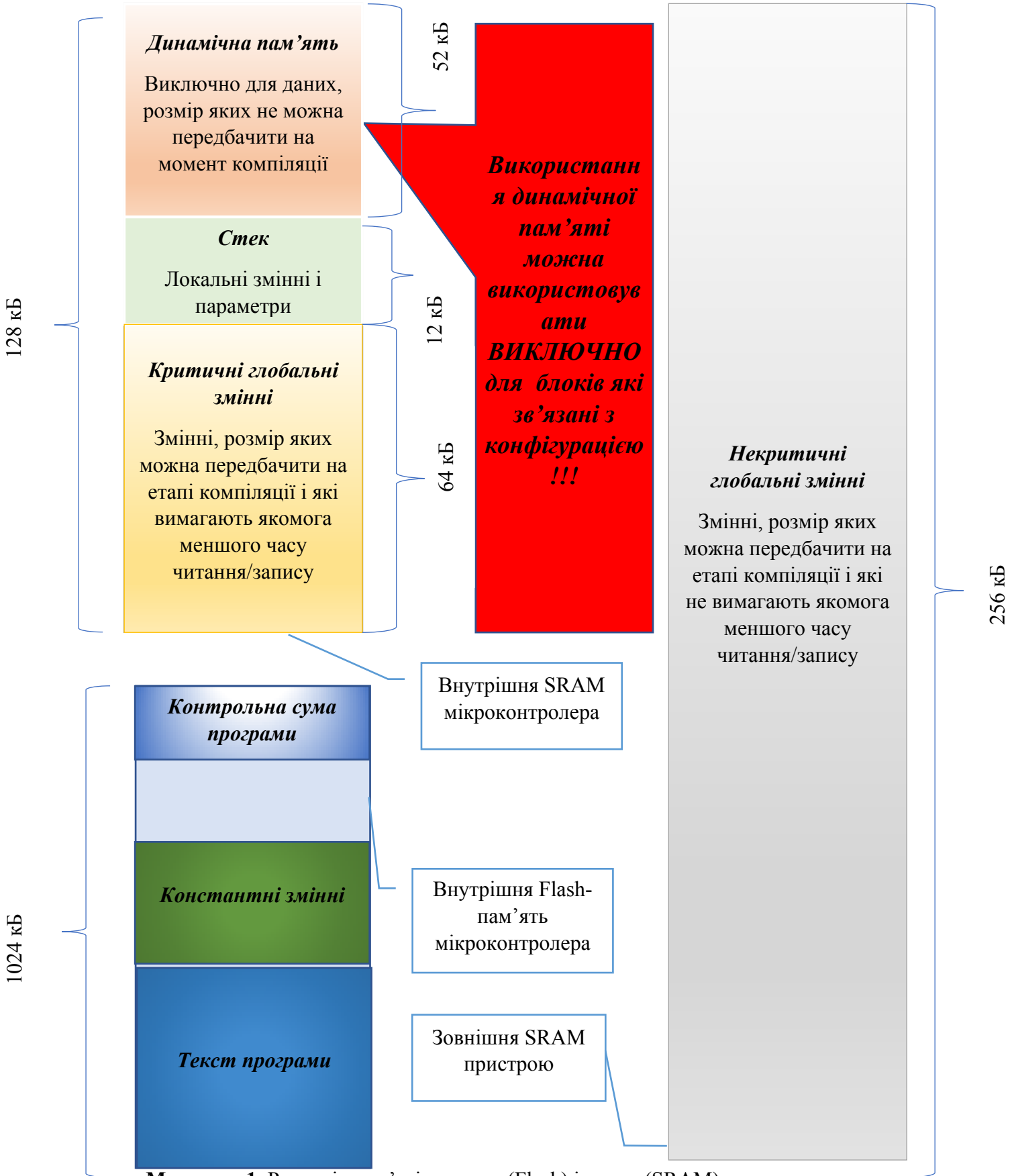
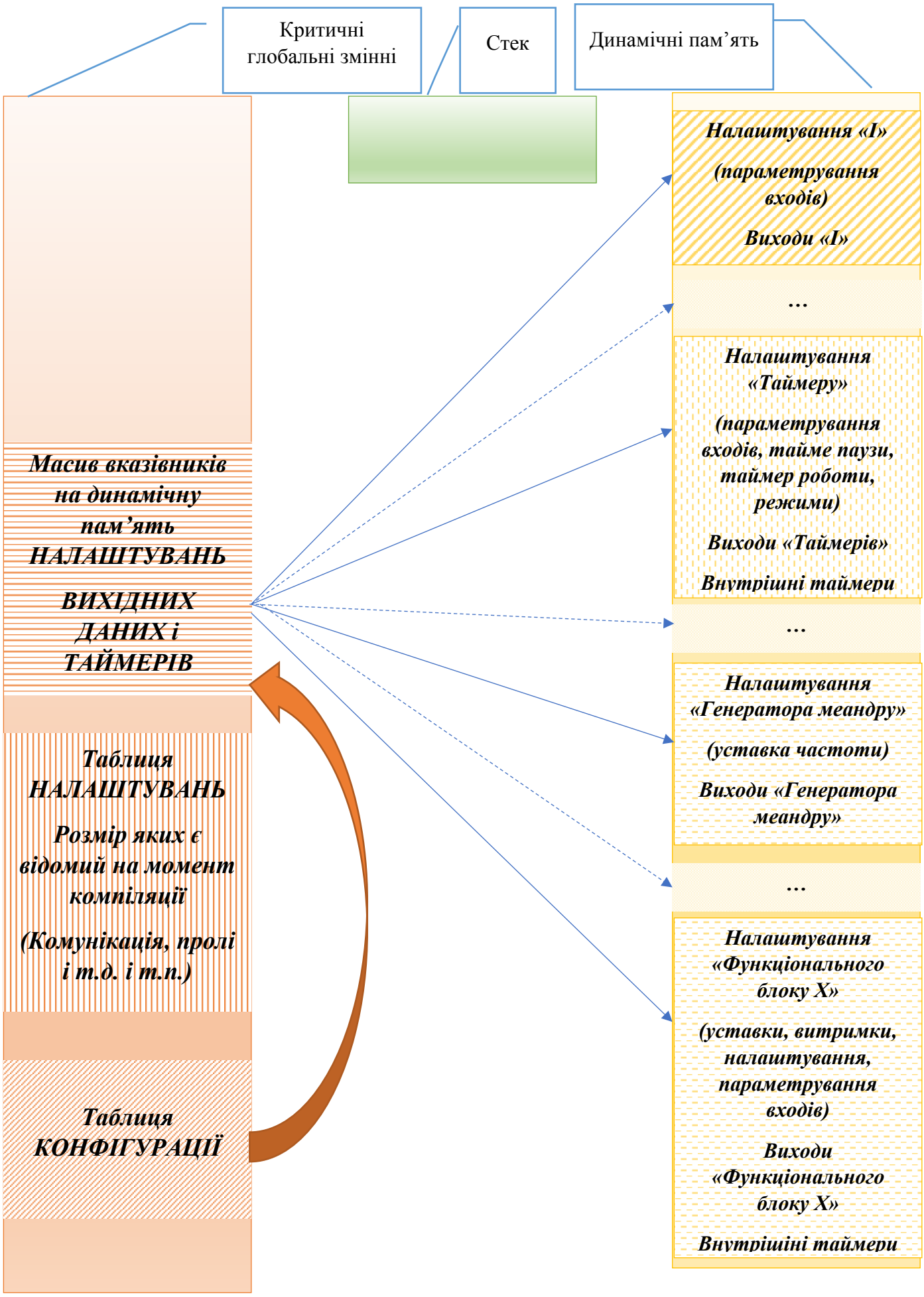
1. Мої пропозиції до організації використання пам’яті для модульного приладу

(з прив’язкою до приладу МРЗС-05Л-ЦС)



Малюнок 1. Розподіл пам’яті програм (Flash) і даних (SRAM) пристрою.



Малюнок 2. Пропозиція по організації конфігурації, налаштувань і вихідних даних логіки пристрою

2.Таблиця конфігурації

*(Об'явлена структура у файлі **type_definition.h**)*

```
typedef struct
{
  uint32_t device_id;                 //Тип пристрою

  uint32_t n_input;                   //Кількість дискретних входів
  uint32_t n_output;                  //Кількість дискретних виходів
  uint32_t n_led;                     //Кількість дискретних світлоіндикаторів

  uint32_t n_and;                     //Кількість елементів "І"
  uint32_t n_or;                      //Кількість елементів "АБО"
  uint32_t n_xor;                     //Кількість елементів "Викл.АБО"
  uint32_t n_not;                     //Кількість елементів "НЕ"
  uint32_t n_timer;                    //Кількість таймерів
  uint32_t n_trigger;                 //Кількість триґерів

  uint32_t n_meandes                  //Кількість генераторів меандру
  uint32_t n_alarm;                   //Кількість блоків сигналізацій


  uint8_t time_config[7+1];       //Час останніх змін уставок-витримок-управління
                              //Останній байт масиву сигналізує мітку звідки зміни були проведені
                              //0 - мінімальні параметри
                              //1 - клавіатура
                              //2 - USB
                              //3 - RS-485

} __CONFIG;
```

3.Формат поля PARAM для визначення, який сигнал заводиться на вхід(входи) функціонального блоку

param - 4-байте слово беззнакове слово(uint32_t)

MSB LSB

4-й байт	3-й байт	2-й байт	1-й байт
ID-функціонального блоку (нумерація іде натуральними числами)	Порядковий номер функціонального блоку (нумерація іде натуральними числами)	Порядковий номер виходу функціонального блоку (нумерація іде натуральними числами)	

*(Константи об'явлені у **const_settings.h**)*

```
#define SFIFT_PARAM_ID          24
#define SFIFT_PARAM_N           16
#define SFIFT_PARAM_OUT          0

#define MASKA_PARAM_ID          ((1 << (8*sizeof(uint32_t) - SFIFT_PARAM_ID )) - 1)
#define MASKA_PARAM_N           ((1 << (SFIFT_PARAM_ID     - SFIFT_PARAM_N  )) - 1)
#define MASKA_PARAM_OUT           ((1 << (SFIFT_PARAM_N      - SFIFT_PARAM_OUT)) - 1)
```

4. Таблиця налаштувань, розміри яких відомо на момент компіляції

*(Об'явлена структура у файлі **type_definition.h**)*

```
typedef struct
{
  //Тиша
  uint32_t ranguvannja_silence;                    //Ранжування сигналізацій
  //Скидання
  uint32_t ranguvannja_reset;                      //Ранжування сигналізацій
  //Тест
  uint32_t ranguvannja_test;                       //Ранжування Тесту

  uint32_t password_1;                             //Пароль для редагування з меню
  uint32_t password_2;                             //Пароль для редагування з меню
  uint32_t timeout_deactivation_password_interface_USB;        //Час деактивації паролю для
редагування з інтерфейсу USB
  uint32_t password_interface_USB;                 //Пароль для редагування з інтерфейсу USB
  uint32_t timeout_deactivation_password_interface_RS485;      //Час деактивації паролю для
редагування з інтерфейсу RS485
  uint32_t password_interface_RS485;               //Пароль для редагування з інтерфейсу RS485

  uint32_t timeout_idle_new_settings;

  //Комунікація
  uint32_t name_of_cell[MAX_CHAR_IN_NAME_OF_CELL];          //І'мя ячейки
  uint32_t volatile address;                       //Адреса
  int32_t speed_RS485;                             //швидкість обміну
                                                   // 0 - 9600
                                                   // 1 - 14400
                                                   // 2 - 19200
                                                   // 3 - 28800
                                                   // 4 - 38400
                                                   // 5 - 57600
                                                   // 6 - 115200
  int32_t pare_bit_RS485;                          //паритет
                                                   // 0 - NONE
                                                   // 1 - ODD
                                                   // 2 - EVEN
  int32_t number_stop_bit_RS485;                   //кількість стоп-біт
                                                   // 0 - 1 stop-bit
                                                   // 1 - 2 stop-bits
  uint32_t time_out_1_RS485;                       //time-out наступного символу = X/10 символу

  int32_t language;                                //мова меню  0= змінна мов не підтримується; 1=RU;
2=UA; 3=EN; 4=KZ; 5=др.

  unsigned char time_setpoints[7+1];               //Час останніх змін уставок-витримок-управління
                                                   //Останній байт масиву сигналізує мітку звідки зміни були
проведені
                                                   //0 - мінімальні параметри
                                                   //1 - клавіатура
                                                   //2 - USB
```

```
//3 - RS-485
} __SETTINGS_FIX;
```

5. Масив вказівників на налаштувань у динамічній пам’яті

*(Перечислення об’явлено у **const_settings.h**)*

```
enum _id_fb
{
  _ID_FB_FIRST_ALL = 1,                                                    /*1*/

    _ID_FB_FIRST_FIX = _ID_FB_FIRST_ALL,                                   /*1*/

      ID_FB_CONTROL_BLOCK = _ID_FB_FIRST_FIX,                              /*1*/

    _ID_FB_LAST_FIX,                                                       /*2*/

    _ID_FB_FIRST_VAR = _ID_FB_LAST_FIX,                                    /*2*/

      _ID_FB_FIRST_VAR_NONE_CHANGED = _ID_FB_FIRST_VAR,                    /*2*/

        ID_FB_INPUT = _ID_FB_FIRST_VAR_NONE_CHANGED,                       /*2*/
        ID_FB_OUTPUT,                                                      /*3*/
        ID_FB_LED,                                                         /*4*/

      _ID_FB_LAST_VAR_NONE_CHANGED,                                        /*5*/

      _ID_FB_FIRST_VAR_CHANGED = _ID_FB_LAST_VAR_NONE_CHANGED,             /*5*/

        ID_FB_AND = _ID_FB_FIRST_VAR_CHANGED,                              /*5*/
        ID_FB_OR,                                                  /*6*/
        ID_FB_XOR,                                                 /*7*/
        ID_FB_NOT,                                                 /*8*/

        ID_FB_TIMER,                                               /*9*/
        ID_FB_TRIGGER,                                             /*10*/

        ID_FB_MEANDER,                                             /*11*/

      _ID_FB_LAST_VAR_CHANGED,                                     /*12*/

    _ID_FB_LAST_VAR = _ID_FB_LAST_VAR_CHANGED,                     /*12*/

  _ID_FB_LAST_ALL = _ID_FB_LAST_VAR                                /*12*/
};
```

*(Макроси об’явлені у **const_settings.h**)*

```
#define NUMBER_FIX_BLOCKS       (_ID_FB_LAST_FIX - _ID_FB_FIRST_FIX)

#define NUMBER_VAR_BLOCKS_NONE_CHANGED  (_ID_FB_LAST_VAR_NONE_CHANGED -
_ID_FB_FIRST_VAR_NONE_CHANGED)
#define NUMBER_VAR_BLOCKS_CHANGED       (_ID_FB_LAST_VAR_CHANGED      -
_ID_FB_FIRST_VAR_CHANGED     )
```

#define NUMBER_VAR_BLOCKS (NUMBER_VAR_BLOCKS_NONE_CHANGED + NUMBER_VAR_BLOCKS_CHANGED)
#define NUMBER_ALL_BLOCKS (NUMBER_FIX_BLOCKS + NUMBER_VAR_BLOCKS)

(Масив об'явлено у ***variables_global.h*** *і* ***variables_external.h****)*
uintptr_t *spca_of_p_prt [NUMBER_VAR_BLOCKS];

6. Макроси

*(Макроси визначені у **macroses.h**)*

```
#define DIV_TO_HIGHER(_N, _M) ((_N / _M) + ((_N % _M) != 0))
```

7. Структури для дискретних входів

*(Перечислення об'явлено у **const_settings.h**)*

```
enum _signals_of_INPUT
{
  INPUT_OUT = 0, /*вихід*/

  INPUT_SIGNALS /*кількість вихідних сигналів*/

};
```

*(Типи об'явлено у **type_definition.h**)*

Структура витримок/таймерів

```
typedef struct
{
  int32_t delay;                                        /*Допуск*/
} __delays_for_INPUT;
```

Структура налаштувань

```
typedef struct
{

  __delays_for_INPUT delay;                             /*Таймери*/
  uint32_t control;                                     /*Управління*/

} __settings_for_INPUT;
```

Структура функціонального елементу

```
typedef struct
{
  __settings_for_INPUT settings;                        /*Налаштування*/

  __delays_for_INPUT delay;
  uint8_t active_state[BLOCK8_SIZE(INPUT_SIGNALS)];     /*Активні сигнали*/
  uint8_t trigger_state[DIV_TO_HIGHER(INPUT_SIGNALS, 8)];./*Спрацьовані (тригерні сигнали)*/
} __LN_INPUT;
```

*(Перечислення об'явлено у **const_menu2_input.h**)*

```
enum __index_ctrl_input_m2 /*перечислення для управління*/
{
  INDEX_CTRL_INPUT_M2_TYPE_SIGNAL = 0, /*0 – Постійний сигнал; 1 – змінний сигнал*/

  MAX_ROW_CTRL_INPUT_M2                 /*Кількість елементів управління*/
};
```

8.Структури для дискретного виходу і світлоіндикатора

*(Перечислення об'явлено у **const_settings.h**)*

```
enum _signals_of_OUTPUT
{
 OUTPUT_OUT = 0,          /*Логічний вихід, без врахування імпульсного режиму*/
 OUTPUT_BOARD,            /*Вихід, який іде на фізичне реле*/

 OUTPUT_SIGNALS           /*Кількість вихідних сигналів*/
};
enum _output_input_signals
{
 OUTPUT_LOGIC_INPUT = 0,  /*логічний вхід*/
 OUTPUT_RESET,            /*вхід сигналу Reset*/
 OUTPUT_MEANDER1,         /*Вхід основного ГПС*/
 OUTPUT_MEANDER2,         /*Вхід допоміжного ГПС*/

 OUTPUT_MAX_NUMBER        /*Кількість входів*/
};

enum _signals_of_LED
{
 LED_OUT = 0,             /*Логічний вихід, без врахування імпульсного режиму*/
 LED_BOARD,                   /*Вихід, який іде на фізичний світлодіод*/

 LED_SIGNALS              /*Кількість вихідних сигналів*/

enum _led_input_signals
{
 LED_LOGIC_INPUT = 0,  /*логічний вхід*/
 LED_RESET,            /*вхід сигналу Reset*/
 LED_MEANDER1,         /*Вхід основного ГПС*/
 LED_MEANDER2,         /*Вхід допоміжного ГПС*/

 LED_MAX_NUMBER        /*Кількість входів*/
};
```

*(Типи об'явлено у **type_definition.h**)*

Структура налаштувань

```
typedef struct
{

 uint32_t control;                                  /*Управління*/
 uint32_t param[OUTPUT_MAX_NUMBER];                 /*Сигнали на входах*/

} __settings_for_OUTPUT;
```

Структура функціонального елементу

```
typedef struct
{
 __settings_for_OUTPUT settings;                                  /*Налаштування*/

 uint8_t active_state[BLOCK8_SIZE(OUTPUT_SIGNALS)];               /*Активні сигнали*/
```

```
  uint8_t trigger_state[DIV_TO_HIGHER(OUTPUT_SIGNALS, 8)];/*Спрацьовані (триґерні сигнали)*/
} __LN_OUTPUT;
```

Структура налаштувань

```
typedef struct
{

  uint32_t control;                                 /*Управління*/
  uint32_t param[LED_MAX_NUMBER];           /*Сигнали на входах*/

} __settings_for_LED;
```

Структура функціонального елементу

```
typedef struct
{
  __settings_for_LED settings;                                    /*Налаштування*/

  uint8_t active_state[DIV_TO_HIGHER(LED_SIGNALS, 8)];   /*Активні сигнали*/
  uint8_t trigger_state[DIV_TO_HIGHER(LED_SIGNALS, 8)];  /*Спрацьовані (триґерні сигнали)*/

} __LN_LED;
```

(Перечислення об’явлено у **const_menu2_output_led.h**)

```
enum __index_ctrl_output_led_m2 /*перечислення для управління*/
{
  INDEX_CTRL_OUTPUT_LED_M2_N_T = 0,    /*0 – Нормальний; 1 – Триґерний*/
  INDEX_CTRL_OUTPUT_LED_M2_C_I,          /*0 – Постійний; 1 – Імпульсний*/
  INDEX_CTRL_OUTPUT_LED_M2_SI_EI,       /*0 – Імп. простий; 1 – Імп. розширений*/
```

*/*Решта елементів перечислення не потрібні для системи обробки логіки елементу*/*

```
_MAX_ROW_CTRL_OUTPUT_LED_M2_BITS_SETTINGS,

  INDEX_CTRL_OUTPUT_LED_M2_MEANDER1 =
_MAX_ROW_CTRL_OUTPUT_LED_M2_BITS_SETTINGS,
  INDEX_CTRL_OUTPUT_LED_M2_MEANDER2,

  MAX_ROW_CTRL_OUTPUT_LED_M2
};
```

9.Структури для елементу стандартної логіки «І»

(Перечислення об’явлено у **const_settings.h**)

```
enum _signals_of_AND
{
  AND_OUT = 0,              /*Логічний вихід*/

  AND_SIGNALS               /*Кількість вихідних сигналів*/

};
```

(Константу об’явлено у **const_settings.h**)

```
#define NUMBER_IN_AND               8
```

(Типи об’явлено у **type_definition.h**)

Структура налаштувань

```
typedef struct
{

  uint32_t param[NUMBER_IN_AND];                    /*Сигнали на входах*/

} __settings_for_AND;
```

Структура функціонального елементу

```
typedef struct
{
  __settings_for_AND settings;                      /*Налаштування*/

  uint8_t active_state[BLOCK8_SIZE(AND_SIGNALS)]; /*Активні сигнали*/
  uint8_t trigger_state[BLOCK8_SIZE(AND_SIGNALS)]; /*Спрацьовані (триґерні сигнали)*/

} __LN_AND;
```

10. Структури для елементу стандартної логіки «АБО»

(Перечислення об’явлено у **const_settings.h**)

```
enum _signals_of_OR
{
  OR_OUT = 0,                          /*Логічний вихід*/

  OR_SIGNALS                           /*Кількість вихідних сигналів*/

};
```

(Константу об’явлено у **const_settings.h**)

```
#define NUMBER_IN_OR                8
```

(Типи об’явлено у **type_definition.h**)

Структура налаштувань

```
typedef struct
{

  uint32_t param[NUMBER_IN_OR];                    /*Сигнали на входах*/

} __settings_for_OR;
```

Структура функціонального елементу

```
typedef struct
{

  __settings_for_OR settings;                      /*Налаштування*/

  uint8_t active_state[BLOCK8_SIZE(OR_SIGNALS)];   /*Активні сигнали*/
  uint8_t trigger_state[BLOCK8_SIZE(OR_SIGNALS)];  /*Спрацьовані (триґерні сигнали)*/

} __LN_OR;
```

11. Структури для елементу стандартної логіки «Викл.АБО»

(Перечислення об’явлено у **const_settings.h**)

```
enum _signals_of_XOR
{
  XOR_OUT = 0,                /*Логічний вихід*/

  XOR_SIGNALS                 /*Кількість вихідних сигналів*/

};
```

(Типи об’явлено у **type_definition.h**)

Структура налаштувань

```
typedef struct
{

  uint32_t param[2];                          /*Сигнали на входах*/

} __settings_for_XOR;
```

Структура функціонального елементу

```
typedef struct
{
  __settings_for_XOR settings;                                  /*Налаштування*/

  uint8_t active_state[BLOCK8_SIZE(XOR_SIGNALS)];  /*Активні сигнали*/
  uint8_t trigger_state[BLOCK8_SIZE(XOR_SIGNALS)]; /*Спрацьовані (триґерні сигнали)*/

} __LN_XOR;
```

12. Структури для елементу стандартної логіки «НЕ»

(Перечислення об’явлено у variables_external.h)

```
enum _signals_of_NOT
{
  NOT_OUT = 0,              /*Логічний вихід*/

  NOT_SIGNALS               /*Кількість вихідних сигналів*/

};
```

(Типи об’явлено у **type_definition.h**)
Структура налаштувань

```
typedef struct
{

  uint32_t param;                                   /*Сигнали на входах*/

} __settings_for_NOT;
```

Структура функціонального елементу

```
typedef struct
{

  __settings_for_NOT settings;                      /*Налаштування*/

  uint8_t active_state[BLOCK8_SIZE(NOT_SIGNALS)];  /*Активні сигнали*/
  uint8_t trigger_state[BLOCK8_SIZE(NOT_SIGNALS)]; /*Спрацьовані (триґерні сигнали)*/

} __LN_NOT;
```

13. Структури для елементу стандартної логіки «БФ-Таймер»

(Перечислення об’явлено у **const_settings.h**)

```
enum _signals_of_TIMER
{
  TIMER_OUT = 0,                    /*логічний вхід*/

  TIMER_SIGNALS                     /*Кількість вихідних сигналів*/

};
enum _output_timer_signals
{
  TIMER_LOGIC_INPUT = 0,            /*логічний вхід*/
  TIMER_RESET,                      /*вхід сигналу Reset*/

  TIMER_MAX_NUMBER                  /*Кількість входів*/
};
```

(Типи об’явлено у **type_definition.h**)

Структура витримок/таймерів

```
typedef struct
{
  int32_t delay_pause;              /*Таймер паузи*/
  int32_t delay_work;               /*Таймер витримки*/

} __delays_for_TIMER;
```

Структура налаштувань

```
typedef struct
{

  uint32_t param;                   /*Сигнали на входах*/
  uint32_t control;                 /*Управління*/
  __delays_for_TIMER delay;

} __settings_for_TIMER;
```

Структура функціонального елементу

```
typedef struct
{

  __settings_for_TIMER settings;                          /*Налаштування*/

  __delays_for_TIMER delay;                               /*Таймери*/
  uint8_t active_state[BLOCK8_SIZE(TIMER_SIGNALS)];       /*Активні сигнали*/
  uint8_t trigger_state[BLOCK8_SIZE(TIMER_SIGNALS)];      /*Спрацьовані (тригерні сигнали)*/

} __LN_TIMER;
```

(Перечислення об’явлено у **const_menu2_timer.h**)

```
enum __index_ctrl_timer_m2 /*перечислення для управління*/
{
  INDEX_CTRL_TIMER_M2_TYPE = 0,    /*0 – Імпульс; 1 – Затримка*/
```

```
  MAX_ROW_CTRL_TIMER_M2             /*Кількість елементів управління*/
};
```

14. Структури для елементу стандартної логіки «Триґер»

(Перечислення об’явлено у **const_settings.h**)

```
enum _signals_of_TRIGGER
{
  TRIGGER_OUT = 0,              /*Логічний вихід (прямий)*/
  TRIGGER_OUT_INV,              /*Логічний вихід (інверсний)*/

  TRIGGER_SIGNALS               /*Кількість вихідних сигналів*/

};
enum _input_signals_of_TRIGGER
{
  INPUT_TRIGGER_SET = 0,            /*Встановлення прямого виходу триґера*/
  INPUT_TRIGGER_RESET,              /*Скидання прямого виходу триґерва*/
  INPUT_TRIGGER_D,                  /*D-вхід*/
  INPUT_TRIGGER_C,                  /*Синхро-вхід*/

  INPUT_TRIGGER_SIGNALS             /*Кількість входів*/

};
```

(Типи об’явлено у **type_definition.h**)

Структура налаштувань

```
typedef struct
{

  uint32_t param[INPUT_TRIGGER_SIGNALS];          /*Сигнали на входах*/


} __settings_for_TRIGGER;
```

Структура функціонального елементу

```
typedef struct
{

  __settings_for_TRIGGER settings;                                  /*Налаштування*/

  uint8_t active_state[BLOCK8_SIZE(TRIGGER_SIGNALS)];   /*Активні сигнали*/
  uint8_t trigger_state[BLOCK8_SIZE(TRIGGER_SIGNALS)];  /*Спрацьовані (тригерні сигнали)*/

} __LN_TRIGGER;
```

15. Структури для елементу функціонального блоку «Генератор періодичних сигналів»

(Перечислення об’явлено у **const_settings.h**)

```
enum _signals_of_MEANDER
{
  MEANDER_OUT = 0,                    /*Логічний вихід*/

  MEANDER_SIGNALS                     /*Кількість вихідних сигналів*/
};
```

(Типи об’явлено у **type_definition.h**)

Структура витримок/таймерів

```
typedef struct
{
  int32_t delay;                      /*Період*/

} __delays_for_MEANDER;
```

Структура налаштувань

```
typedef struct
{

  __delays_for_MEANDER delay;         /*Витримки*/

} __settings_for_MEANDER;
```

Структура функціонального елементу

```
typedef struct
{

  __settings_for_MEANDER settings;                          /*Налаштування*/

  __delays_for_MEANDER delay;                               /*Таймери*/
  uint8_t active_state[BLOCK8_SIZE(MEANDER_SIGNALS)];       /*Активні сигнали*/
} __LN_MEANDER;
```

Моя пропозиція (дискусійна!) щодо алгоритму опрацювання логіки всіх функціональних блоків у конкретний часовий відрізок

Умова завершення опрацьовування всіх функціональних блоків
- опрацьовано всі функціональні блоки
- вхідні сигнали між I-тою ітерацією і (I - 1)-тою ітерацією для всіх функціональних блоків не змінилися
- максимальні кількість ітерацій не перевищує критично встановлене значення

У випадку, якщо кількість ітерацій перевищила максимально-допустиму кількість ітерацій у конкретний часовий відрізок, то встановлюється сигнал «Помилка параметрування внутрішніх зв'язків».

Другою умовою дострокового припинення пошуку стаціонарного стану є випадок, якщо стан для будь-яких J-ої ітерації і K-тої ітерації **ВСІ** стани сигналів повторилися. У такому випадку ітераційний процес також завершується і виставляється сигнал «Помилка параметрування внутрішніх зв'язків»

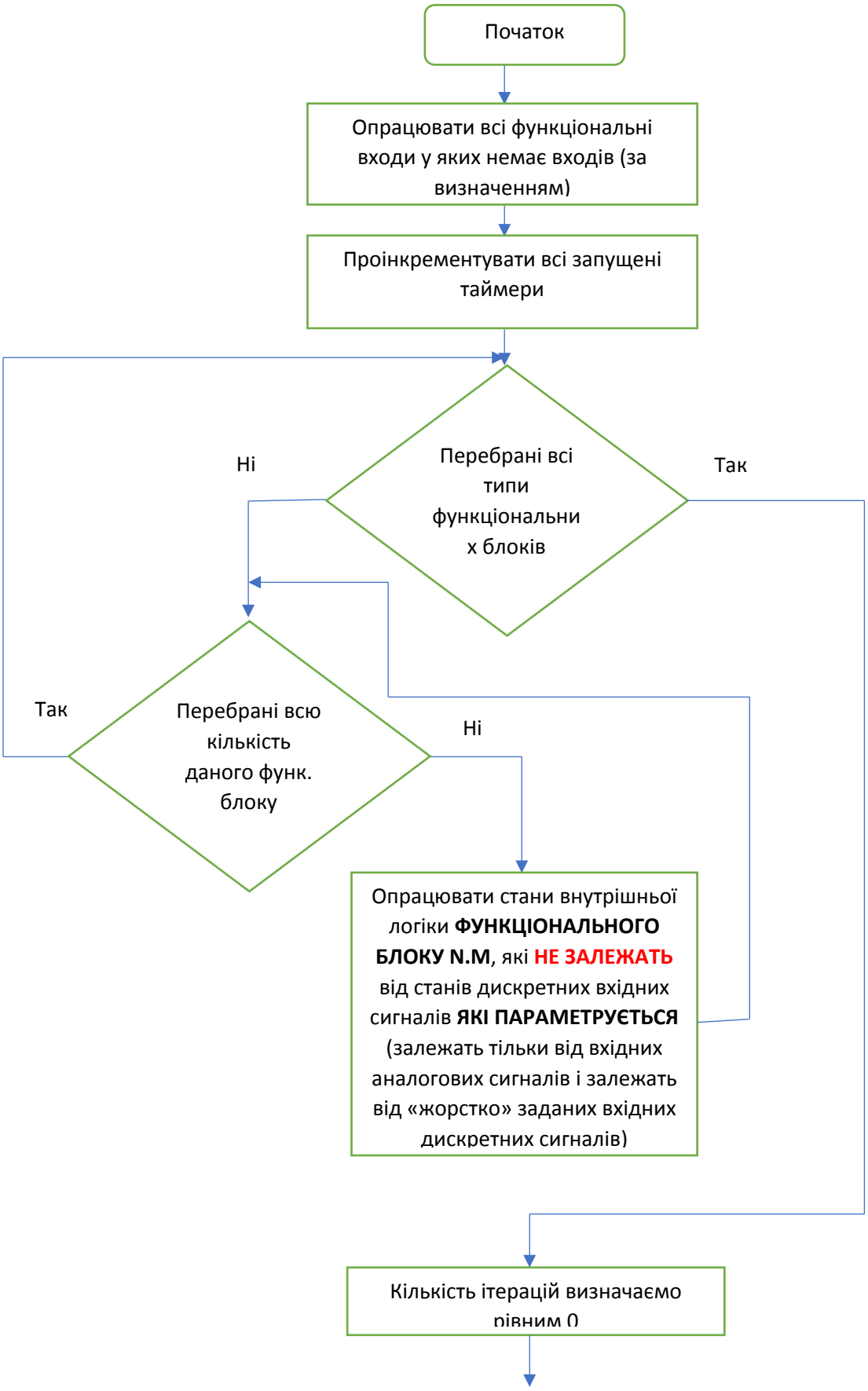
Для реалізації визначення того, що існує, а бо не існує два проміжні стани у перехідному процесі, коли всі сигнали повторюються, а також для фіксації ситуації що встановився стаціонарний стан (між двома останніми ітераціями не відбувалося зміни станів) у ДИНАМІЧНІЙ пам'яті виділяється масив розміром «МАКСИМАЛЬНА КІЛЬКІСТЬ ІТЕРАЦІЙ»х«КІЛЬКІСТЬ 32-БІТНИХ СЛІВ ДЛЯ ВСІХ СИГНАЛІВ ВИБАНОЇ КОНФІГУРАЦІЇ[1]»

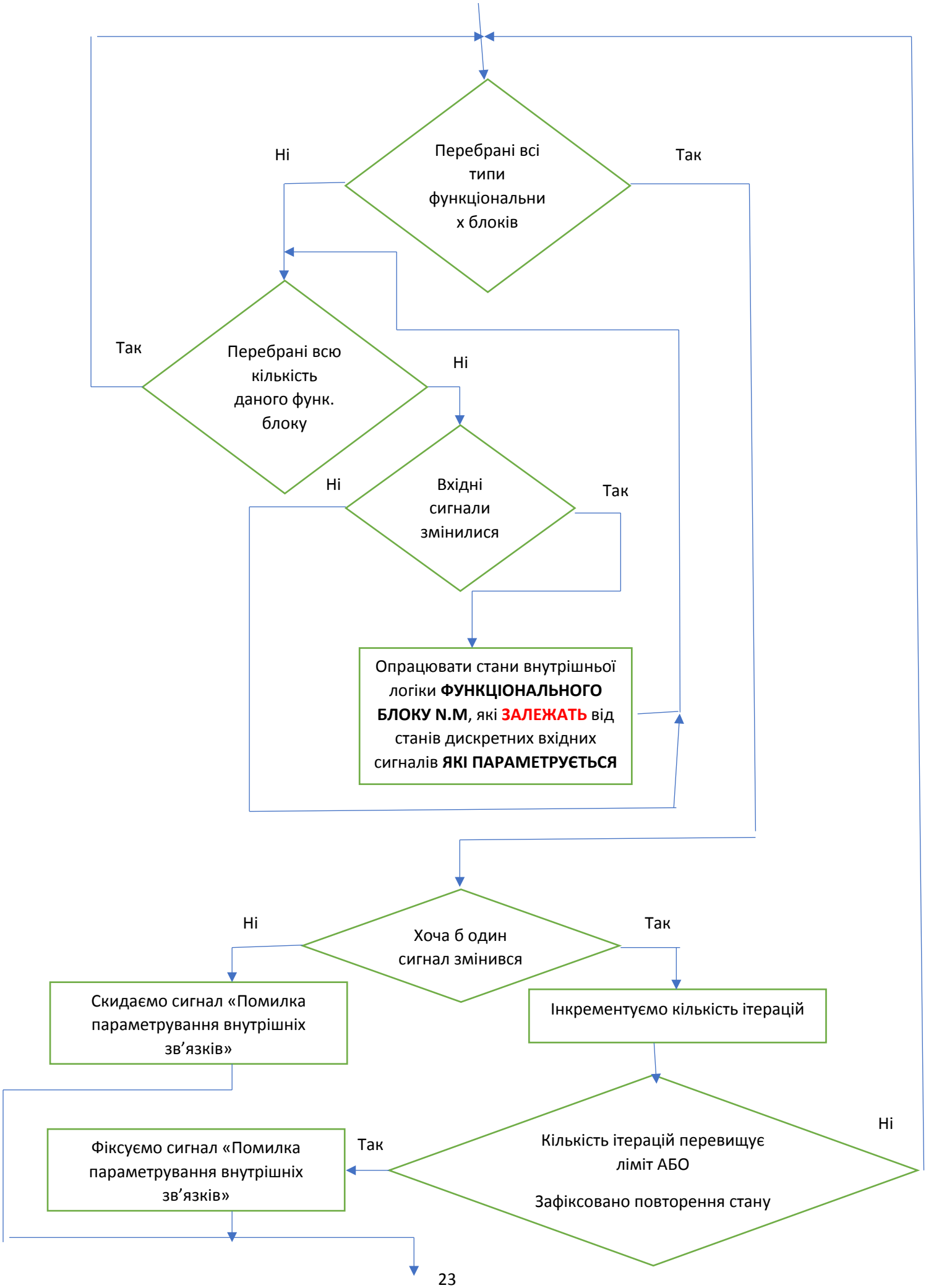
N – Максимальна кількість ітерацій
M – максимальна кількість 32-бітних слів для всіх сигналів вибраної конфігурації

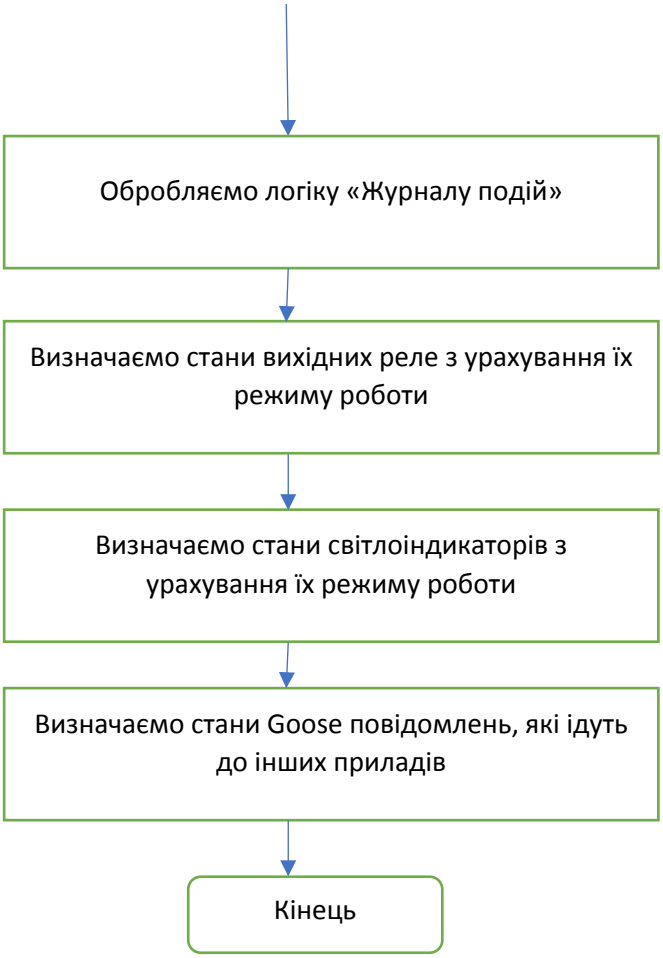
	LSB			HSB
0 ітерація	0 - слово	1 - слово	…	(M – 1) - слово
1 ітерація	0 - слово	1 - слово	…	(M – 1) - слово
		…		
(I - 1) ітерація	0 - слово	1 - слово	…	(M – 1) - слово
I ітерація	0 - слово	1 - слово	…	(M – 1) - слово
(I + 1) ітерація	0 - слово	1 - слово	…	(M – 1) - слово
		…		
N ітерація	0 - слово	1 - слово	…	(M – 1) - слово

[1] Оскільки загальна кількість сигналів залежить від вибраної конфігурації, то цей масив буде мати розмір залежний від вибору конфігурації, а значить буде розміщений у динамічний пам'яті. Вказівник на нього буде встановлюватися при старті системи і змінюватися при зміні конфігурації. Так само у визначеній змінній буде визначено при старті системи і перевизначено після зміни конфігурації величина «кількість 32-бітних слів для всіх сигналів вибраної конфігурації».

Блок схема обробки всіх функціональних блоків




Перебрані всі типи функціональни х блоків
Ні
Так
Перебрані всю кількість даного функ. блоку
Так
Ні
Вхідні сигнали змінилися
Ні
Так
Опрацювати стани внутрішньої логіки **ФУНКЦІОНАЛЬНОГО БЛОКУ N.M**, які **ЗАЛЕЖАТЬ** від станів дискретних вхідних сигналів **ЯКІ ПАРАМЕТРУЄТЬСЯ**
Хоча б один сигнал змінився
Ні
Так
Скидаємо сигнал «Помилка параметрування внутрішніх зв’язків»
Інкрементуємо кількість ітерацій
Фіксуємо сигнал «Помилка параметрування внутрішніх зв’язків»
Так
Кількість ітерацій перевищує ліміт АБО
Зафіксовано повторення стану
Ні


Обробляємо логіку «Журналу подій»
Визначаємо стани вихідних реле з урахування їх режиму роботи
Визначаємо стани світлоіндикаторів з урахування їх режиму роботи
Визначаємо стани Goose повідомлень, які ідуть до інших приладів
Кінець