Для включения библиотеки для работы с Flash сделать следующее.

1) Включить в проект скомпилированные библиотеки:

canv_lib.lib
flash_lib.lib

2) Библиотека для CAN обновлена. Изменения для интерфейса коснулись аргументов у функции

int CAN_Init(CAN_TypeDef *can_ref, uint32_t BTR_reg);

Теперь в зависимости от тактовой частоты нужно определить значения настраиваемого регистра CAN->BTR и передать его вторым аргументом. Все остальное остается как прежде. Старую верстку с исходниками – выкинуть.

3) В коде, там, где заведен список регистрируемых переменных, добавить последней запись:

```
#include "canv.h"
#include "flash_lib.h"

const typeRegistrationRec RegistrationRec[] = {
  /*{указатель на IVar, размер IVar, указатель на Callback, id_IVar, доступ}*/
  ................,
  ................,
  {&VarCAN_FlashFragment, 0x80008, CallbackCAN_Flash, 14, CAN_AFLG_NOOFFSET}
};
```

Чтобы подготовить образ программы, нужно из полученного .hex файла создать бинарный с помощью программки Hex2Bin_wCRC.exe . Иметь в виду, что перед компиляцией настройку стартового адреса надо выбирать в зависимости от того, в каком блоке предназначается работать: 0x8000000 или 0x8080000

Библиотеки выставлены уже в скомпилированном виде, потому, что мною при компиляции предусмотрены некоторые тонкие моменты: нужная сегментация, выбор нужных ключей, определений и т.д.

---------- все ----------

Для тех, кто захочет кроме радио программировать Flash напрямую через CAN

Процесс программирования ведется через переменную VarID = 14.

У этой переменной две области:

Offs=0x00000 .. 0x7FFFF – запись или чтение фрагмента Flash по адресу 0x8080000+Offs, если программа крутится в 1-м блоке; или 0x8000000+Offs , если – во 2-м блоке
Offs=0x80000 – 8-ми байтовый блок управления и статуса:

```
typedef union {
  uint8_t   bb[8];
  struct {
    uint32_t  size : 24;
    uint32_t  cmd :   8;
```

```
    uint32_t  crc;
    } Ctrl;
  struct {
    int8_t Status;
    uint8_t CurrentBlock;
    uint8_t PrefBlock;
    uint8_t ResetSrc;
    } Info;
}typeUniVarCANFlashFragment;
```

здесь
cmd – команда на действия:

FLASH_CMD_ERASE_SECTORS_2	0x02	стирание всех секторов выше 0x8080000, если программа крутится в 1-м блоке; или всех секторов ниже 0x8080000, если – во 2-м блоке
FLASH_CMD_CHECK_CRC_IMAGE_N	0x04*	проверка CRC содержимого блока
FLASH_CMD_CHECK_VALID_IMAGE_N	0x06*	проверка валидности содержимого блока с исп. вн. CRC
FLASH_CMD_FIX_VALID_IMAGE_N	0x08*	подсчет и запись внутр. CRC
FLASH_CMD_SET_PREF_BLOCK_N	0x0A*	установка предпочитаемого стартового блока
FLASH_CMD_DO_COPY_AND_GO	0x0C	копирование 2-го блока в 1-й и рестарт
FLASH_CMD_RESTART	0x0E	просто рестарт

* – номер целевого блока в 0-м бите

В Status отражается статус выполнения команды. Если команда выполнена корректно, то в него заносится код выполненной команды. Если с ошибкой – то в старшем (7-м) бите будет выставлена единица. В случае успешной записи фрагмента во Flash статус = 0x10, при ошибке = 0x90.

size, crc – размер и контр. сумма (CRC32, описание в даташите на STM32) проверяемого образа, требуется для выполнения команд FLASH_CMD_CHECK_CRC_IMAGE_N, FLASH_CMD_DO_COPY_AND_GO.

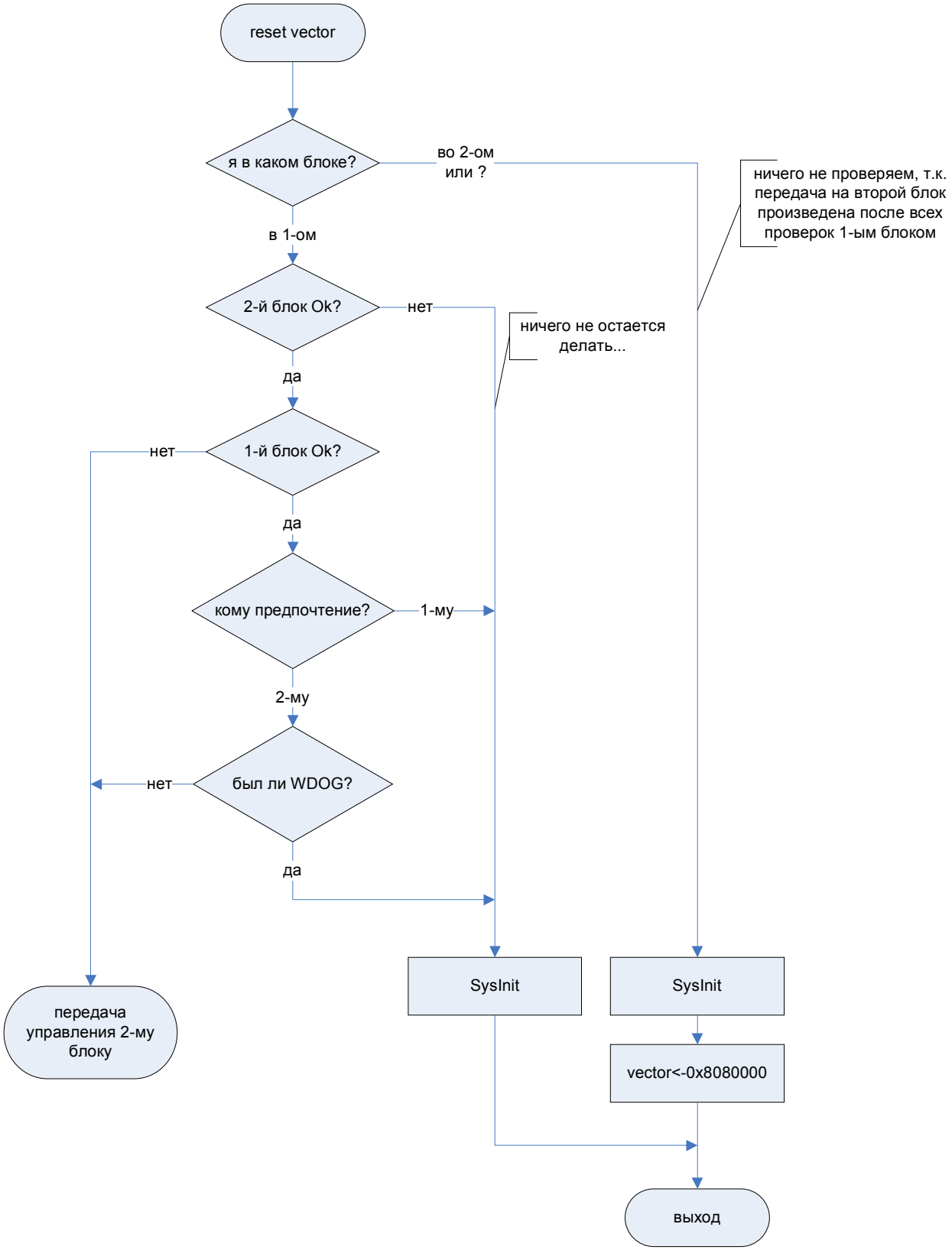
CurrentBlock – номер блока в котором сейчас крутится программа

PrefBlock – установленный предпочитаемый блок

ResetSrc – источник сброса, содержимое регистра RCC_CSR[31:24]

---------- ----------

Для желающих иметь представление об алгоритме выбора стартового блока, привожу структуру того, с чего начинается выполнение кода


reset vector
я в каком блоке?
во 2-ом
или ?
ничего не проверяем, т.к. передача на второй блок произведена после всех проверок 1-ым блоком
в 1-ом
2-й блок Ok?
нет
ничего не остается делать...
да
1-й блок Ok?
нет
да
кому предпочтение?
1-му
2-му
был ли WDOG?
нет
да
передача управления 2-му блоку
SysInit
SysInit
vector<-0x8080000
выход