# Введение

Акционерное общество Научно-технический центр «Модуль» более 30 лет успешно работает на российском рынке наукоемких технологий. С 1995 года компания создает высокопроизводительные процессорные ядра и аналогово-цифровые системы-на-кристалле. Сегодня, благодаря высокой квалификации сотрудников и самому современному оснащению, НТЦ «Модуль» разрабатывает и производит аппаратуру управления и контроля самых современных авиационных и космических систем, аппаратно-программные решения в области нейронных сетей, в том числе в части обработки видеопотока и изображений, навигации, связи, обнаружения и распознавания объектов, занимается контрактным выполнением ОКР и НИР.

НТЦ модуль занимается:

* Проектированием и производством вычислительных модулей, систем управления (бортовая и авиационная аппаратура);
* Проектирование интегральных микросхем (услуги микроэлектронного дизайна);
* Внедрение нейронных сетей и отечественной компонентной базы в современные автоматизированные комплексы различных направлений: от навигации до беспилотных автомобилей и робототехники;
* Производство и проектирование систем распознавания и анализа видеоизображений;
* Разработка и производство навигационного оборудования, в том числе высокоточного GNSS позиционирования;
* Разработка СФ-блоков.

В собственности компании вычислительные мощности и технологии, позволяющие проводить наукоемкие исследования и разработки. НТЦ «Модуль» является лицензиатом консорциумов HDMI® и DCP LLC®, имеет аттестованное и оснащенное современным оборудованием сборочное производство, обеспечивающее мелкосерийный выпуск встраиваемых компьютеров и модулей.

# Исходные данные

Необходимо разработать программный модуль, предназначенный для преобразования BSD-файла (JTAG, Boundary Scan) согласно необходимой методике тестирования выбранного устройства, сконфигурированной специалистом в SVF-файл, пригодный для передачи в программу, проводящую граничное тестирование исследуемой платы или её составной части, в данном случае OpenOCD.

# Описание полученного задания

JTAG (Joint Test Action Group) — это стандартный интерфейс для тестирования и отладки цифровых устройств. Был разработан для упрощения диагностики и проверки сложных электронных систем, таких как интегральные схемы (ИС) и печатные платы (PCB). JTAG обеспечивает средство для взаимодействия с внутренними структурами этих устройств через стандартный физический интерфейс и набор команд. JTAG определяется стандартом IEEE 1149.1, который описывает протокол и электрические характеристики для тестирования и отладки через специализированный интерфейс. На рисунке 1 приведена обобщённая структура микросхемы, оборудованная интерфейсом JTAG.

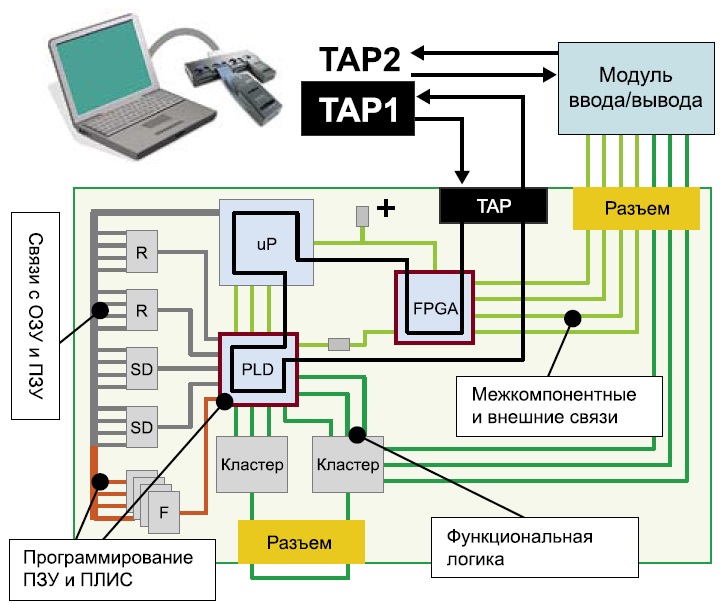


Рисунок – Обобщённая структура микросхемы, оборудованная интерфейсом JTAG

BSD-файла описываем структуру и тип выводов микросхемы, а также описание регистров Boundary Scan предназначенных для проведения тестирования. Пример BSD-файла приведён в листинге 1.

Листинг 1 – Пример BSD-файла

1. entity SIMPLE\_IC is
2. generic (PHYSICAL\_PIN\_MAP : string);
3. port (
4. LED: out bit\_vector(0 to 7);
5. BTN: in bit\_vector(0 to 7);
6. TMS: in bit;
7. TDI: in bit;
8. TCK: in bit;
9. TDO: out bit;
10. VDD: linkage bit;
11. VSS: linkage bit;
12. NC: linkage bit\_vector(0 to 9)
13. );
14. use STD\_1149\_1\_2001.all;
15. attribute COMPONENT\_CONFORMANCE of SIMPLE\_IC : entity is "STD\_1149\_1\_2001";
16. attribute PIN\_MAP of SIMPLE\_IC : entity is PHYSICAL\_PIN\_MAP;
17. constant PLCC20:PIN\_MAP\_STRING:=
18. "LED: (1,2,3,4), " &
19. "BTN: (8,7,6,5), " &
20. "TMS: 9, " &
21. "TDI: 10, " &
22. "TCK: 11, " &
23. "TDO: 12, " &
24. "VDD: 13, " &
25. "VSS: 14, " &
26. "NC: (15,16,17,18,19,20) ";
27. attribute TAP\_SCAN\_MODE of TMS : signal is true;
28. attribute TAP\_SCAN\_IN of TDI : signal is true;
29. attribute TAP\_SCAN\_CLOCK of TCK : signal is (10.0e3, BOTH);
30. attribute TAP\_SCAN\_OUT of TDO : signal is true;
31. attribute INSTRUCTION\_LENGTH of SIMPLE\_IC : entity is 8;
32. attribute INSTRUCTION\_OPCODE of SIMPLE\_IC : entity is
33. "IDCODE (00000001)," &
34. "EXTEST (00000010)," &
35. "BYPASS (11111111)," &
36. "SAMPLE (00000100) ";
37. attribute INSTRUCTION\_CAPTURE of SIMPLE\_IC : entity is "00000001";
38. attribute IDCODE\_REGISTER of SIMPLE\_IC : entity is
39. "0000" & -- код ревизии или чего-нибудь типа того
40. "1010101001010101" & -- код модели микросхемы hAA55
41. "00000000001" & -- код производителя (соответствует AMD)
42. "1"; -- единица по стандарту IEEE1149.1
43. attribute REGISTER\_ACCESS of SIMPLE\_IC : entity is
44. "DEVICE\_ID (IDCODE)," &
45. "BYPASS (BYPASS)," &
46. "BOUNDARY (EXTEST, SAMPLE)";
47. attribute BOUNDARY\_LENGTH of SIMPLE\_IC : entity is 8;
48. attribute BOUNDARY\_REGISTER of SIMPLE\_IC : entity is
49. "0 (BC\_1, LED(0), output2, X)," &
50. "1 (BC\_1, LED(1), output2, X)," &
51. "2 (BC\_1, LED(2), output2, X)," &
52. "3 (BC\_1, LED(3), output2, X)," &
53. "4 (BC\_1, BTN(0), input, X)," &
54. "5 (BC\_1, BTN(1), input, X)," &
55. "6 (BC\_1, BTN(2), input, X)," &
56. "7 (BC\_1, BTN(3), input, X)" ;
57. end SIMPLE\_IC;

SVF-файл (Serial Vector Format) используется в контексте JTAG (Joint Test Action Group) для описания последовательности команд, которые могут быть использованы для программирования или тестирования цифровых устройств. Формат SVF предоставляет стандартный способ представления тестовых векторов для выполнения различных операций через JTAG-интерфейс. SVF-файл содержит инструкции и данные, которые описывают последовательность команд для взаимодействия с JTAG-совместимыми устройствами. Этот формат позволяет автоматизировать процесс тестирования и программирования, предоставляя универсальный способ управления устройствами через JTAG. Пример SVF-файла приведён в листинге 2.

Листинг – Пример SVF-файла

1. !Begin Test Program
2. TRST OFF;
3. !Disable Test Reset line
4. ENDIR IDLE;
5. !End IR scans in IDLE
6. ENDDR IDLE;
7. !End DR scans in IDLE
8. HIR 8 TDI (00);
9. !8-bit IR header
10. HDR 16 TDI (FFFF) TDO (FFFF) MASK (FFFF);
11. !16-bit DR header
12. TIR 16 TDI (0000);
13. !16-bit IR trailer
14. TDR 8 TDI (12);
15. !16-bit DR trailer
16. SIR 8 TDI (41);
17. !8-bit IR scan
18. SDR 32 TDI (ABCD1234) TDO (11112222);
19. !32-bit DR scan
20. STATE DRPAUSE;
21. !Go to stable state DRPAUSE
22. RUNTEST 100 TCK ENDSTATE IRPAUSE;
23. !RUNBIST for 100 TCKs
24. !End Test Program

OpenOCD (Open On-Chip Debugger) – это бесплатный инструмент с открытым исходным кодом для отладки и программирования микроконтроллеров и микропроцессоров через интерфейсы, такие как JTAG и SWD (Serial Wire Debug). OpenOCD поддерживает множество различных архитектур и устройств и часто используется в сочетании с различными IDE (Integrated Development Environment) и инструментами разработки.

# Решение поставленной задачи

## *Модуль работы с BSD-файлом*

Листинг кода формирователя итогового SVF-файла приведён на листинге 3, листинге 4, листинге 5.

Листинг 3 – Заголовочный файл библиотеки pininfo.hpp

1. #**pragma once**
2. #include <string>
3. #include <vector>
4. #include <unordered\_map>
5. // Структура для хранения информации о порте и ячейках
6. **class** BsdlPins{
7. **public**:
9. **class** PinInfo {
10. **public**:
11. **enum** **class** StatePin {
12. high,
13. low,
14. z,
15. x
16. };
17. std::string pin; // номер физического пина, 0 для не выведенных пинов
18. std::string label; // название физического пина
19. std::string pin\_type; // тип ячейки in, out, inout
20. **unsigned** **int** In; // номер ячейки ввода
21. **unsigned** **int** Out; // номер ячейки вывода
22. **unsigned** **int** Config; // ячейка управления
23. std::string function; // <function> ячейки BS
24. **bool** turnOff; // при каком значении в ячейки происходит отключение драйвера 1 или 0
26. std::string stateOff; // состояние выходного отключенного драйвера z, 1 (high), 0 (low)
27. std::string safeState; // безопасное значение ячейки X, 1 (high), 0 (low)
28. };
30. **void** loadBsdl(std::string filename){
32. // Читаем данные из файла и записываем в переменную content
33. std::string content = readFile(filename);
34. // Получаем даныне о пинах и заносим данные в переменную ping (vector)
35. pins = parseBSDFile(content);
36. // Получаем данные об имени пина и его номере
37. std::unordered\_map<std::string, std::string> pinMap = parsePinMap(content);
39. // Получаем данные об имени пина и его типе
40. std::unordered\_map<std::string, std::string> pinTypes = parsePinTypes(content);
41. // Устанавливаем связь номеров пинов и их типов
42. mapPinNumbersAndTypes(pins, pinMap, pinTypes);
43. // Удаляем дублирующиеся пины
44. pins = removeDuplicatePins(pins);
45. }
46. // Функция для вывода информации о пинах
47. **void** printPinInfo(std::ostream &os=std::cout);
48. **protected**:
49. // Защищенные методы для выполнения операций
50. // Функция для чтения файла и возврата его содержимого в виде строки
51. **static** std::string readFile(**const** std::string& filename);
52. // Функция для преобразования str в bool
53. **static** **bool** stringToBool(**const** std::string& str);
54. // Функция для парсинга файла .bsd и извлечения информации о пинах
55. **static** std::vector<PinInfo> parseBSDFile(**const** std::string& content);
56. // Функция для парсинга строки с описанием пина
57. **static** PinInfo parsePinInfo(**const** std::string& line);
58. // Функция для парсинга строк с номерами пинов
59. **static** std::unordered\_map<std::string, std::string> parsePinMap(**const** std::string& content);
60. // Функция для парсинга строк с типами пинов
61. **static** std::unordered\_map<std::string, std::string> parsePinTypes(**const** std::string& content);
62. // Функция для удаления дублирующихся пинов и переноса значения Cell In из дубликата в первый пин
63. **static** std::vector<PinInfo> removeDuplicatePins(**const** std::vector<PinInfo>& pins);
64. // Функция для установки связи номеров пинов и их типов
65. **static** **void** mapPinNumbersAndTypes(std::vector<PinInfo>& pins, **const** std::unordered\_map<std::string,
66. std::string>& pinMap, **const** std::unordered\_map<std::string, std::string>& pinTypes);
67. **private**:
68. std::vector<PinInfo> pins;
69. };

Листинг 4 – Файл реализации библиотеки pininfo.cpp

1. #include <iostream>
2. #include <fstream>
3. #include <regex>
4. #include <string>
5. #include <unordered\_map>
6. #include "pininfo.hpp"
7. // Функция для преобразования str в bool
8. **bool** BsdlPins::stringToBool(**const** std::string& str) {
9. **int** value = std::stoi(str);
10. **return** value != 0;
11. }
12. // Функция для чтения файла и возврата его содержимого в виде строки
13. std::string BsdlPins::readFile(**const** std::string& filename) {
14. std::ifstream file(filename);
15. **if** (!file.is\_open()) {
16. **throw** std::runtime\_error("Could not open file");
17. }
18. **return** std::string((std::istreambuf\_iterator<**char**>(file)), std::istreambuf\_iterator<**char**>());
19. }
20. // Функция для парсинга строки с описанием пина
21. BsdlPins::PinInfo BsdlPins::parsePinInfo(**const** std::string& line) {
22. std::regex pinRegex(R"(\s\*(\d+)\s\*\((\w+),\s\*(\S\*),\s\*(\w+),?\s\*(\w\*),?\s\*(\S\*),?\s\*(\w\*).\s\*(\w\*))");
23. std::smatch match;
24. PinInfo pinInfo;
25. **if** (std::regex\_search(line, match, pinRegex)) {
26. pinInfo.label = match[3].str();
27. pinInfo.function = match[4].str();
28. pinInfo.safeState = match[5].str();
29. match[7].str().empty() ? pinInfo.turnOff = 0 : pinInfo.turnOff = stringToBool(match[7].str());
30. pinInfo.stateOff = match[8].str();
31. **if** (match[4].str() == "INPUT") {
32. pinInfo.In = std::stoi(match[1].str());
33. match[6].str().empty() ? pinInfo.Config = 0 : pinInfo.Config = std::stoi(match[6].str());
34. pinInfo.Out = 0;
35. } **else** **if** (match[4].str() == "OUTPUT3") {
36. pinInfo.Out = std::stoi(match[1].str());
37. match[6].str().empty() ? pinInfo.Config = 0 : pinInfo.Config = std::stoi(match[6].str());
38. pinInfo.In = 0;
39. } **else** {
40. pinInfo.In = 0; pinInfo.Out = 0; pinInfo.Config = 0;
41. }
42. }
43. **return** pinInfo;
44. }
45. // Функция для парсинга строк с номерами пинов
46. std::unordered\_map<std::string, std::string> BsdlPins::parsePinMap(**const** std::string& content) {
47. std::unordered\_map<std::string, std::string> pinMap;
48. std::regex pinEntryRegex(R"(\s\*(\w+)\s\*:\s\*(\d+),)");
49. **auto** lines\_begin = std::sregex\_iterator(content.begin(), content.end(), pinEntryRegex);
50. **auto** lines\_end = std::sregex\_iterator();
51. **for** (std::sregex\_iterator i = lines\_begin; i != lines\_end; ++i) {
52. std::smatch match = \*i;
53. std::string label = match[1].str();
54. std::string pin = match[2].str();
55. pinMap[label] = pin;
56. }
57. **return** pinMap;
58. }
59. // Функция для парсинга строк с типами пинов
60. std::unordered\_map<std::string, std::string> BsdlPins::parsePinTypes(**const** std::string& content) {
61. std::unordered\_map<std::string, std::string> pinTypes;
62. std::regex pinTypeRegex(R"(\s\*(\w+)\s\*:\s\*(\w+)\s\*(\w\*);)");
63. **auto** lines\_begin = std::sregex\_iterator(content.begin(), content.end(), pinTypeRegex);
64. **auto** lines\_end = std::sregex\_iterator();
65. **for** (std::sregex\_iterator i = lines\_begin; i != lines\_end; ++i) {
66. std::smatch match = \*i;
67. std::string label = match[1].str();
68. std::string pin\_type = match[2].str();
69. pinTypes[label] = pin\_type;
70. }
71. **return** pinTypes;
72. }
73. // Функция для парсинга файла .bsd и извлечения информации о пинах
74. std::vector<BsdlPins::PinInfo> BsdlPins::parseBSDFile(**const** std::string& content) {
75. std::vector<PinInfo> pins;
76. std::regex lineRegex(R"((\d+\s\*\(BC\_\d+,.\*\)))");
77. **auto** lines\_begin = std::sregex\_iterator(content.begin(), content.end(), lineRegex);
78. **auto** lines\_end = std::sregex\_iterator();
79. **for** (std::sregex\_iterator i = lines\_begin; i != lines\_end; ++i) {
80. std::smatch match = \*i;
81. PinInfo pin = parsePinInfo(match.str());
82. pins.push\_back(pin);
83. }
84. **return** pins;
85. }
86. // Функция для удаления дублирующихся пинов и переноса значения Cell In из дубликата в первый пин
87. std::vector<BsdlPins::PinInfo> BsdlPins::removeDuplicatePins(**const** std::vector<PinInfo>& pins) {
88. std::unordered\_map<std::string, PinInfo> pinMap;
89. **for** (**const** **auto**& pin : pins) {
90. **if** (pinMap.find(pin.pin) == pinMap.end()) {
91. pinMap[pin.pin] = pin;
92. } **else** {
93. **if** (pin.function == "INPUT" && pinMap[pin.pin].function == "OUTPUT3") {
94. pinMap[pin.pin].In = pin.In;
95. }
96. }
97. }
99. std::vector<PinInfo> result;
100. **for** (**const** **auto**& pair : pinMap) {
101. result.push\_back(pair.second);
102. }
103. **return** result;
104. }
105. // Функция для установки связи номеров пинов и их типов
106. **void** BsdlPins::mapPinNumbersAndTypes(std::vector<PinInfo>& pins, **const** std::unordered\_map<std::string,
107. std::string>& pinMap, **const** std::unordered\_map<std::string, std::string>& pinTypes) {
108. **for** (**auto**& pin : pins) {
109. **if** (pinMap.find(pin.label) != pinMap.end()) {
110. pin.pin = pinMap.at(pin.label);
111. } **else** {
112. pin.pin = "0"; // Используем 0 для ячеек BS, к которым не привязаны пины
113. }
114. **if** (pinTypes.find(pin.label) != pinTypes.end()) {
115. pin.pin\_type = pinTypes.at(pin.label);
116. } **else** {
117. pin.pin\_type = "unknown"; // Используем "unknown" для неизвестных типов
118. }
119. }
120. }
121. // Функция для вывода информации о пинах
122. **void** BsdlPins::printPinInfo(std::ostream &os) {
123. **for** (**const** **auto**& pin : pins) {
124. os << "Pin: " << pin.pin
125. << ", Port Name: " << (pin.label.empty() ? "\*" : pin.label) // condition ? true\_value : false\_value
126. << ", Pin type: " << pin.pin\_type
127. << ", Function: " << pin.function
128. << ", Cell In: " << pin.In
129. << ", Cell Out: " << pin.Out
130. << ", Cell Config: " << pin.Config
131. << ", Disable Value: " << pin.turnOff
132. << ", Safe State: " << (pin.safeState.empty() ? "N/A" : pin.safeState)
133. << ", State Off: " << (pin.stateOff.empty() ? "N/A" : pin.stateOff)
134. << std::endl;
135. }
136. }

Листинг 5 – Файл, работающий с библиотекой pininfo.hpp

1. #include <iostream>
2. #include <string>
3. #include <unordered\_map>
4. #include <vector>
5. #include <string>
6. # include "pininfo.hpp"
7. **int** main(**int** argc, **char**\* argv[]) {
8. **if** (argc < 2) {
9. std::cerr << "Usage: " << argv[0] << " <bsd-file> " << std::endl;
10. **return** 1;
11. }
13. // Получаем имя файла
14. std::string filename = argv[1];
15. BsdlPins BsdlPins;
16. // Вызываем методы загрузки и обработки bsd
17. BsdlPins.loadBsdl(filename);
18. // Выводим информацию о пинах
19. BsdlPins.printPinInfo();
20. **return** 0;
21. }

## *Модуль формирования задачи тестирования*

Листинг кода формирователя задачи тестирования приведён на листинге 6, листинге 7, листинге 8.

Листинг 6 – Заголовочный файл библиотеки svf-calc.hpp

1. #**pragma once**
2. #include <string>
3. #include <vector>
4. #include <nlohmann/json.hpp>
5. **using** json = nlohmann::json;
6. **class** JsonForm {
7. **public**:
8. **void** fileForm(**int** argc, **char** \*argv[]);
9. **private**:
10. std::vector<std::vector<std::string>> pinsList;
11. std::vector<std::vector<std::string>> writeStatusList;
12. std::vector<std::vector<std::string>> readStatusList;
13. std::string filenameBSD;
14. **void** parsingArguments(**int** argc, **char** \*argv[], std::vector<std::vector<std::string>> &pinsList,
15. std::vector<std::vector<std::string>> &writeStatusList, std::vector<std::vector<std::string>> &readStatusList,
16. std::string &filenameBSD);
17. **bool** writeJsonToFile(**const** std::string& filename, **const** nlohmann::json& jsonObject);
18. std::string replaceExtension(**const** std::string& filename, **const** std::string& oldExt, **const** std::string& newExt);
19. json createJsonObject(**const** std::vector<std::vector<std::string>>& pinsList,
20. **const** std::vector<std::vector<std::string>>& writeStatusList,
21. **const** std::vector<std::vector<std::string>>& readStatusList);
22. };

Листинг 7 – Файл реализации библиотеки svf-calc.hpp

1. #include <iostream>
2. #include <fstream>
3. #include "svf-calc.hpp"
4. **void** JsonForm::fileForm(**int** argc, **char** \*argv[]){
5. // Парсинг аргументов
6. parsingArguments(argc, argv, pinsList, writeStatusList, readStatusList, filenameBSD);
7. // Создание имени файла json
8. std::string filename = replaceExtension(filenameBSD, ".bsd", "\_test.json");
9. // Создание json объекта
10. json jsonArray = createJsonObject(pinsList, writeStatusList, readStatusList);
11. // Запись json в файл
12. **if** (writeJsonToFile(filename, jsonArray)) {
13. std::cout << "Создан файл: " << filename << std::endl;
14. }
15. **return**;
16. }
17. // Функция для парсинга аргументов командной строки
18. **void** JsonForm::parsingArguments(**int** argc, **char** \*argv[], std::vector<std::vector<std::string>> &pinsList,
19. std::vector<std::vector<std::string>> &writeStatusList, std::vector<std::vector<std::string>> &readStatusList,
20. std::string &filenameBSD) {
21. std::vector<std::string> pins;
22. std::vector<std::string> writeStatus;
23. std::vector<std::string> readStatus;
24. **for** (**int** i = 1; i < argc; ++i) {
25. **if** (std::string(argv[i]) == "--pins" && i + 1 < argc) {
26. **if** (!pins.empty()) {
27. pinsList.push\_back(pins);
28. writeStatusList.push\_back(writeStatus);
29. readStatusList.push\_back(readStatus);
30. pins.clear();
31. writeStatus.clear();
32. readStatus.clear();
33. }
34. std::string pinsStr = argv[++i];
35. size\_t pos = 0;
36. **while** ((pos = pinsStr.find(',')) != std::string::npos) {
37. pins.push\_back(pinsStr.substr(0, pos));
38. pinsStr.erase(0, pos + 1);
39. }
40. pins.push\_back(pinsStr);
41. } **else** **if** (std::string(argv[i]) == "--write" && i + 1 < argc) {
42. std::string writeStr = argv[++i];
43. size\_t pos = 0;
44. **while** ((pos = writeStr.find(',')) != std::string::npos) {
45. writeStatus.push\_back(writeStr.substr(0, pos));
46. writeStr.erase(0, pos + 1);
47. }
48. writeStatus.push\_back(writeStr);
49. } **else** **if** (std::string(argv[i]) == "--read" && i + 1 < argc) {
50. std::string readStr = argv[++i];
51. size\_t pos = 0;
52. **while** ((pos = readStr.find(',')) != std::string::npos) {
53. readStatus.push\_back(readStr.substr(0, pos));
54. readStr.erase(0, pos + 1);
55. }
56. readStatus.push\_back(readStr);
57. } **else** **if** (std::string(argv[i]) == "--filename" && i + 1 < argc) {
58. filenameBSD = argv[++i];
59. }
60. }
61. **if** (!pins.empty()) {
62. pinsList.push\_back(pins);
63. writeStatusList.push\_back(writeStatus);
64. readStatusList.push\_back(readStatus);
65. }
66. }
67. // Функция создания и записи в файл
68. **bool** JsonForm::writeJsonToFile(**const** std::string& filename, **const** nlohmann::json& jsonObject) {
69. std::ofstream file(filename);
70. **if** (!file.is\_open()) {
71. std::cerr << "Файл: " << filename << " не может быть открыт" << std::endl;
72. **return** **false**;
73. }
74. file << jsonObject.dump(4); // параметр 4 задает отступы для красивого форматирования
75. file.close();
76. **return** **true**;
77. }
78. // Функция замены расширения файла
79. std::string JsonForm::replaceExtension(**const** std::string& filename, **const** std::string& oldExt, **const** std::string& newExt) {
80. size\_t pos = filename.rfind(oldExt);
81. **if** (pos != std::string::npos && pos == filename.length() - oldExt.length()) {
82. **return** filename.substr(0, pos) + newExt;
83. } **else** {
84. std::cerr << "Некорректное расширение файла: " << filename << std::endl;
85. exit(1);
86. }
87. }
88. // Функция создания json объекта
89. json JsonForm::createJsonObject(**const** std::vector<std::vector<std::string>>& pinsList,
90. **const** std::vector<std::vector<std::string>>& writeStatusList,
91. **const** std::vector<std::vector<std::string>>& readStatusList) {
92. json jsonArray = json::array();
93. **for** (size\_t i = 0; i < pinsList.size(); ++i) {
94. json jsonObject;
95. jsonObject["pins"] = pinsList[i];
96. jsonObject["write"] = writeStatusList[i];
97. jsonObject["read"] = readStatusList[i];
98. jsonArray.push\_back(jsonObject);
99. }
100. **return** jsonArray;
101. }

Листинг 8 – Файл, работающий с библиотекой svf-calc.hpp

1. #include <iostream>
2. #include "svf-calc.hpp"
3. **using** json = nlohmann::json;
4. // Основная функция
5. **int** main(**int** argc, **char** \*argv[]) {
6. **if** (argc < 2) {
7. std::cerr << "Общая структура: --filename \"name\_file\_BSD.bsd\" --pins \"pin\_name\_1, ..., pin\_name\_n\" --write \"pin\_status\_1, ..., pin\_status\_n\" --read \"pin\_status\_1, ..., pin\_status\_n\"" << std::endl;
8. **return** 1;
9. } **else** **if** (argc < 4) {
10. std::cerr << "Структура пинов: --pins \"pin\_name\_1, ..., pin\_name\_n\" --write \"pin\_status\_1, ..., pin\_status\_n\" --read \"pin\_status\_1, ..., pin\_status\_n\"" << std::endl;
11. **return** 1;
12. }
13. JsonForm JsonForm;
14. JsonForm.fileForm(argc, argv);
15. **return** 0;
16. }

## *Модуль формирования итогового SVF-файла*

Листинг кода формирователя итогового SVF-файла приведён на листинге 9, листинге 10, листинге 11.

Листинг 9 – Заголовочный файл библиотеки svf-generator.hpp

1. #**pragma once**
2. #include <string>
3. #include <unordered\_set>
4. **class** StateArg {
5. **public**:
6. // Публичный метод для запуска формирования SVF
7. **void** formation\_svf(**int** argc, **char** \*argv[]);
8. **private**:
9. // Параметры для формирования SVF
10. std::string filename\_bsd = "NO FILE";
11. std::string filename\_json = "NO FILE";
12. std::string trst\_state = "OFF";
13. std::string endir\_state = "IDLE";
14. std::string enddr\_state = "IDLE";
15. // Приватные методы
16. **void** print\_usage();
17. **bool** is\_valid\_state(**const** std::string& state, **const** std::string valid\_states[], size\_t count);
18. **bool** has\_extension(**const** std::string& filename, **const** std::unordered\_set<std::string>& validExtensions);
19. **void** parse\_arguments(**int** argc, **char** \*argv[]);
20. std::string replaceExtension(**const** std::string& filename, **const** std::string& oldExt, **const** std::string& newExt);
21. **void** createFile(**char** \*argv[]);
22. };

Листинг 10 – Файл реализации библиотеки svf-generator.hpp

1. #include "svf-generator.hpp"
2. #include <iostream>
3. #include <fstream>
4. #include <vector>
5. #include <unordered\_set>
6. #include <getopt.h>
7. // Публичная метод запускающий privat методы
8. **void** StateArg::formation\_svf(**int** argc, **char** \*argv[]){
9. // Парсинг аргументов
10. parse\_arguments(argc, argv);
11. // Создание файла svf
12. createFile(argv);
13. }
14. // Функция вывода help
15. **void** StateArg::print\_usage() {
16. std::cout << "Usage: program [options]\n"
17. << "Options:\n"
18. << " -b, --bsdl Add a BSDL-file\n"
19. << " -j, --json Add a JSON-file\n"
20. << " -t, --trst Set the TRST state (ON, OFF, z, ABSENT)\n"
21. << " -i, --endir Set the ENDIR state (IRPAUSE, DRPAUSE, RESET, IDLE)\n"
22. << " -d, --enddr Set the ENDDR state (IRPAUSE, DRPAUSE, RESET, IDLE)\n"
23. << " -h, --help Show this help message\n";
24. }
25. // Функция проверки корректности id введённого в CLI аргументов
26. **bool** StateArg::is\_valid\_state(**const** std::string& state, **const** std::string valid\_states[], size\_t count) {
27. **for** (size\_t i = 0; i < count; ++i) {
28. **if** (state == valid\_states[i]) {
29. **return** **true**;
30. }
31. }
32. **return** **false**;
33. }
34. // Функция проверки расширения файла
35. **bool** StateArg::has\_extension(**const** std::string& filename, **const** std::unordered\_set<std::string>& validExtensions) {
36. size\_t pos = filename.rfind('.');
37. **if** (pos != std::string::npos && pos != filename.length() - 1) {
38. std::string fileExt = filename.substr(pos + 1);
39. **return** validExtensions.find(fileExt) != validExtensions.end();
40. }
41. **return** **false**;
42. }
43. // Функция парсинга аргументов и вывода записанных аргументов в консоль
44. **void** StateArg::parse\_arguments(**int** argc, **char** \*argv[]){
45. // Инициализация вспомагательных переменных
46. **int** option\_index = 0;
47. **int** c;
48. // Инициализация доступных аргументов
49. **static** **struct** option long\_options[] = {
50. {"bsdl", required\_argument, 0, 'b'},
51. {"json", required\_argument, 0, 'j'},
52. {"trst", required\_argument, 0, 't'},
53. {"endir", required\_argument, 0, 'i'},
54. {"enddr", required\_argument, 0, 'd'},
55. {"help", no\_argument, 0, 'h'},
56. {0, 0, 0, 0}
57. };
58. // Инициализация доступных состояний аргументов
59. **const** std::string trst\_states[] = {"ON", "OFF", "z", "ABSENT"};
60. **const** std::string endir\_enddr\_states[] = {"IRPAUSE", "DRPAUSE", "RESET", "IDLE"};
61. // Цикл проверки всех переданных аргументов
62. **while** ((c = getopt\_long(argc, argv, "b:j:t:i:d:h", long\_options, &option\_index)) != -1) {
63. **switch** (c) {
64. **case** 'b':
65. filename\_bsd = optarg;
66. **if** (!has\_extension(filename\_bsd,{"bsd","bsdl"})) {
67. std::cerr << "Неверное расширение BSDL-файла (.bsd)\n";
68. abort();
69. }
70. **break**;
71. **case** 'j':
72. filename\_json = optarg;
73. **if** (!has\_extension(filename\_json, {"json"})) {
74. std::cerr << "Неверное расширение JSON-файла (.json)\n";
75. abort();
76. }
77. **break**;
78. **case** 't':
79. trst\_state = optarg;
80. **if** (!is\_valid\_state(trst\_state, trst\_states, 4)) {
81. std::cerr << "Неверное состояние --trst. Возможные состояния ON, OFF, z, or ABSENT.\n";
82. abort();
83. }
84. **break**;
85. **case** 'i':
86. endir\_state = optarg;
87. **if** (!is\_valid\_state(endir\_state, endir\_enddr\_states, 4)) {
88. std::cerr << "Неверное состояние --endir. Возможные состояния IRPAUSE, DRPAUSE, RESET, or IDLE.\n";
89. abort();
90. }
91. **break**;
92. **case** 'd':
93. enddr\_state = optarg;
94. **if** (!is\_valid\_state(enddr\_state, endir\_enddr\_states, 4)) {
95. std::cerr << "Неверное состояние --enddr. Возможные состояния IRPAUSE, DRPAUSE, RESET, or IDLE.\n";
96. abort();
97. }
98. **break**;
99. **case** 'h':
100. print\_usage();
101. **return**;
102. **default**:
103. abort();
104. }
105. }
106. // Проверка на наличие BSD-файла и JSON-файла
107. **if**((filename\_bsd == "NO FILE")||(filename\_json == "NO FILE")){
108. std::cout << "Необходимо указать имя BSDL-файла и JSON-файла" << std::endl;
109. abort();
110. }
111. // Вывод записанных аргументов
112. std::cout << "\nBSDL-file: " << filename\_bsd << "\n";
113. std::cout << "JSON-file: " << filename\_json << "\n";
114. std::cout << "TRST state: " << trst\_state << "\n";
115. std::cout << "ENDIR state: " << endir\_state << "\n";
116. std::cout << "ENDDR state: " << enddr\_state << "\n";
117. }
118. // Функция замены расширения файла
119. std::string StateArg::replaceExtension(**const** std::string& filename, **const** std::string& oldExt, **const** std::string& newExt) {
120. size\_t pos = filename.rfind(oldExt);
121. **if** (pos != std::string::npos && pos == filename.length() - oldExt.length()) {
122. **return** filename.substr(0, pos) + newExt;
123. } **else** {
124. std::cerr << "Некорретное раcширение файла: " << filename << std::endl;
125. exit(1);
126. }
127. }
128. // Функция создающая файл и заполняющая его в соотвествии с json
129. **void** StateArg::createFile(**char** \*argv[]){
130. // Создание имени файла с расширением svf
131. std::string filename\_svf = replaceExtension(filename\_json, ".json", ".svf");
132. // Открытие файла для записи
133. std::ofstream svfFile(filename\_svf);
134. // Запись данных в файл (длина регистра 5 bit для STM32F1)
135. svfFile << "! Начать программу тестирования\n"
136. "TRST " << trst\_state << ";\n";
137. svfFile << "ENDIR "<< endir\_state << ";\n";
139. svfFile << "ENDDR "<< enddr\_state << ";\n";
140. **unsigned** **int** count = 1;
141. **unsigned** **int** instr\_len = 5;
142. std::string EXTEST = "00000";
143. **unsigned** **int** bound\_len = 139;
145. std::vector<**char**> pin\_tdi(bound\_len, 0);
147. std::vector<**char**> pin\_tdo(bound\_len, 0);
148. std::vector<**char**> pin\_mask(bound\_len, 0);
150. **for**(**int** i = 0; i < count; i++){
151. svfFile << "TIR " << instr\_len << " TDI (" << EXTEST << ")\n";
152. // svfFile << "SDR " << bound\_len << " TDI " << pin\_tdi << " TDO " << pin\_tdo << " MASK (" << pin\_mask << ");\n";
154. svfFile << "RUNTEST 100 TCK ENDSTATE IDLE;\n";
155. }
156. // Закрытие файла
157. svfFile.close();
158. // Успешное завершение программы
159. std::cout << "\nФайл " << filename\_svf << " успешно создан." << std::endl;
160. }

Листинг 11 – Файл, работающий с библиотекой svf-generator.hpp

1. #include <iostream>
2. #include "svf-generator.hpp"
3. #include "pininfo.hpp"
4. **int** main(**int** argc, **char**\* argv[]) {
5. // Получаем имя файла
6. std::string filename = argv[1];
7. BsdlPins BsdlPins;
8. // Вызываем методы загрузки и обработки bsd
9. BsdlPins.loadBsdl(filename);
10. StateArg StateArg;
11. // Парсим аргументы введённые при запуске и создаём SVF-файл
12. StateArg.formation\_svf(argc, argv);
14. **return** 0;
15. }

# Список литературы

* + - 1. Интерфейс JTAG? — Это очень просто URL: <https://habr.com/ru/articles/190012/>;
      2. Разглядывая JTAG: \*.bsdl своими руками URL: <https://habr.com/ru/articles/660795/>;
      3. IEEE Std. 1149.1 - Standard Test Access Port and Boundary-Scan Architecture URL: <https://grouper.ieee.org/groups/1149/1/>;
      4. JTAG-тест: мифы и реальность URL: <https://tech-e.ru/2011_6_57.php>;
      5. IEEE Std 1149.1 (JTAG) Testability URL: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.ti.com/lit/an/ssya002c/ssya002c.pdf;
      6. SVF File URL: <https://www.xjtag.com/about-jtag/svf-files/>;
      7. SVF SERIAL VECTOR FORMAT SPECIFICATION JTAG│BOUNDARY SCAN URL: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.asset-intertech.com/wp-content/uploads/2020/09/svf-serial-vector-format-specification-jtag-boundary-scan-revision-e.pdf.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_ 2024г.



подпись обучающегося расшифровка подписи дата