



МОНИТОРИНГ СОСТОЯНИЯ АППАРАТНО-ПРОГРАММНОГО СТЕНДА ДЛЯ ОТРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Н. И. Синёв, Д. М. Астахов, И. М. Воронов, Д. И. Яцкевич

Санкт-Петербургский государственный университет аэрокосмического приборостроения

В статье рассматривается проблема диагностики вычислительных и аппаратно-программных устройств. Представлено программное обеспечение, позволяющее в автоматическом режиме диагностировать оборудование уникальной научной установки "Аэрокосмический стенд SpaceWire для исследования, сертификации и тестирования". Информация о состоянии аппаратно-программного узла выводится в графическом виде после синтаксического разбора.

Ключевые слова: графическая программа, управляющий компьютер, последовательный порт, диагностика, журнал событий, linux, minicom, Qt, C++.

Для цитирования:

Синёв, Н. И. Мониторинг состояния аппаратно-программного стенда для отработки программного обеспечения / Н. И. Синёв, Д. М. Астахов, И. М. Воронов, Д. И. Яцкевич // Системный анализ и логистика. – 2023. – № 3(37). – с. 44 – 60. DOI: 10.31799/2077-5687-2023-3-44-60.

MONITORING THE STATUS OF THE HARDWARE AND SOFTWARE STAND FOR TESTING SOFTWARE

N. I. Sinyov, D. M. Astakhov, I. M. Voronov, D. I. Yatskevich

St. Petersburg State University of Aerospace Instrumentation

The article deals with the problem of diagnostics of computing and hardware-software devices. The software is presented, which allows to automatically diagnose the equipment of the unique scientific stand "Aerospace stand SpaceWire for research, certification and testing". Information about the state of the hardware and software node is displayed graphically after parsing.

Keywords: graphics program, control computer, serial port, diagnostics, event log, linux, minicom, Qt, C++.

For citation:

Sinyov, N. I. Monitoring the status of the hardware and software stand for testing software / N. I. Sinyov, D. M. Astakhov, I. M. Voronov, D. I. Yatskevich // System analysis and logistics. – 2023. – № 3(37). – p. 44 – 60. DOI: 10.31799/2077-5687-2023-3-44-60.

Введение

Разработка аппаратно-программных комплексов является важной задачей, позволяющей продемонстрировать новую технологию перед этапом ее внедрения. При этом наиболее важной частью разработки является диагностика и мониторинг состояния комплектующих.

В большинстве случаев компьютерные программы скрывают процесс своего выполнения, выдавая только конечный результат. В таких ситуациях, без возможности отлаживать программный код, не представляется возможным отследить причину ошибочного результата или почему результат выполнения не совпадает с ожидаемым выводом. Для решения данной проблемы необходимо сохранять информацию о произошедших событиях на устройствах в журнал, который в дальнейшем можно проанализировать и выявить причину сбоя программы.

1. Проблемы диагностики последовательных устройств

Если иметь дело только с одним несвязным устройством, то ручная диагностика не будет занимать много времени. В другом случае, когда имеется большой набор из вычислительных узлов, которые связаны между собой через маршрутизаторы, диагностика каждого устройства по отдельности занимает много времени сотрудников лаборатории или предприятия, а также аппаратных и финансовых средств.

Еще одной важной проблемой является то, что обычному пользователю бывает сложно



обойтись без графического программного обеспечения. Если инженер может обойтись консольными приложениями, то менее квалифицированному сотруднику или стажеру, как правило, не хватает знаний для диагностики устройства через командную строку. Для того, чтобы решить эту проблему необходим программный комплекс с удобным графическим окружением, которое бы могло собирать, обрабатывать, систематизировать и представлять в наглядном виде пользователю всю информацию о состоянии устройств в уникальной научной установки “Аэрокосмический стенд SpaceWire для исследования, сертификации и тестирования” (Далее - УНУ АССИСТ) [1].

2. Алгоритм извлечения журнала событий

УНУ АССИСТ включает в себя отладочные модули с подключённым процессором Салют-ЭЛ240М1 [2] и МС-30SF6ЕМ-6U [3], а также коммутатор МСК-02REM-3U [4]. Каждое из данных устройств соединено с управляющим компьютером Raspberry Pi 3 Model B+ (Далее - Raspberry Pi) [5] через UART (Далее - последовательный порт), с помощью которого и производится диагностика устройства. Между собой устройства соединены интерфейсом SpaceWire [6], через который осуществляется передача пакетов данных.

Подключение к управляющему компьютеру производилось через сетевой протокол SSH. В сети Raspberry Pi имел название `pi@stand-rpi`.

Так как на Raspberry Pi установлена операционная система Raspberry Pi OS [7], всем подключенным внешним устройствам присваивается свой файл TTY (teletype), который лежит по пути `/dev/`. В нашем случае устройствам файлы внешних последовательных устройств имели имена `/dev/ttyUSBn`, где `n` - номер устройства от 0 до 2 (Далее - `ttyUSB`).

Начиная с пункта 2.3 в статье описываются команды терминала, которые входят в стандартный набор `linux`-команд.

Для реализации графического программного обеспечения и алгоритма извлечения информации с устройств был использован язык программирования C++ с фреймворком Qt версии 5.12.10 (Далее - Qt) [8].

На момент написания статьи на УНУ АССИСТ только вычислительный модуль Салют-ЭЛ240М1 имел возможность хранить журналы внутренних событий. Поэтому дальнейшая работа проводилась лишь с этим устройством. На этом устройстве установлена операционная система Linux. Название устройства в операционной системе - `mscm02`.

Первой задачей являлась разработка алгоритма извлечения и обработки журналов событий устройства через последовательный порт. Есть несколько способов для решения данной задачи, рассмотрим их ниже.

2.1. Программа на Си/C++

Языки программирования Си и C++ имеют несколько библиотек для работы с последовательным портом [9]. Для настройки последовательного соединения с внешним устройством используется библиотека `termios.h`.

Создав объект структуры `struct termios`, можно получить текущие настройки последовательного порта через функцию `tcgetattr()`:

```
tcgetattr(fileno(stdin), &termios_object);
```

Для записи новых настроек в терминал используется функция `tcsetattr()`:

```
tcsetattr(fileno(stdin), TCSANOW, &termios_object);
```

Когда последовательный порт настроен, необходимо получить файловый дескриптор файла `ttyUSB` для дальнейшего вызова таких функций, как `read()` и `write()` из библиотеки `unistd.h`.

Сохранение файлового дескриптора в переменную `port_fd`:

```
int port_fd = open ("/dev/ttyUSB", O_RDONLY);
```



Чтение информации из порта 20 символов и сохранение их в массив символов *buffer*:

```
read(port_fd, &buffer, 20);
```

Запись 20 символов из массива *buffer* в последовательный порт:

```
write(port_fd, buffer, 20);
```

Недостатки способа:

- Функция *read()* считывает определенное количество символов из строки. Если строка оказывается меньше указанной в параметре длины, то после считывания символов 13 и 10 из таблицы ASCII, символ возврата каретки и символа перехода на новую строку соответственно, функция *read()* начинает выдавать неопределенное поведение, перекидывая ошибки на считывание следующей строки. Количество символов в каждой строке является разным, а количество строк достигает до 2500. Предугадать программным кодом количество символов в строке перед ее непосредственным чтением невозможно.
- Устройство, на которое мы отправляли команды через функцию *write()*, не воспринимало их без дополнительных скриптов и обработчиков.

2.2. Программа на Qt C++

Фреймворк Qt добавляет в язык программирования C++ библиотеку для работы с последовательным портом *QSerialPort* [10]. Создав указатель на объект данного класса, для начала необходимо вызвать метод *setPortName()*, в параметр которого вводится название необходимого файла ttyUSB:

```
QSerialPort* serial_port = new QSerialPort();  
serial_port->setPortName("/dev/ttyUSB");
```

В дальнейшем можно задавать настройки работы с последовательным портом, например, задать скорость передачи информации через метод *setBaudRate()* или поменять количество читаемых бит с помощью метода *setDataBits()*:

```
serial_port->setBaudRate(QSerialPort::Baud115200);  
serial_port->setDataBits(QSerialPort::Data8);
```

Для дальнейшей работы с последовательным портом необходимо вызвать метод *open()* у объекта класса *QSerialPort* для открытия устройства:

```
serial_port->open(QIODevice::ReadWrite)
```

Для записи данных в последовательный порт используется метод *write()*, а для считывания информации - метод *readAll()*:

```
serial_port->write("something");  
serial_port->readAll();
```

Преимущества способа

- Если сравнивать с библиотеками *termios.h* и *unistd.h*, то класс *QSerialPort* является более удобным инструментом для работы с последовательным портом.

Недостатки способа:

- Аналогично программе на Си/C++ из предыдущего пункта программа, написанная на Qt C++, требует обработчик с обратной стороны связи.

2.3. Работа с последовательным терминалом

2.3.1. Передача команд через ТТУ

Чтобы получить информацию об устройстве, нужно вводить команды в терминал. С



помощью команды перенаправления вывода “>” можно посылать текст на последовательное устройство через ttyUSB. Например:

```
uname -a > /dev/ttyUSB0
```

Однако команда выполняется на управляющем компьютере, а результат выполнения будет перенаправлен в ttyUSB. Данную проблему можно решить командой echo.

```
echo "uname -a" > /dev/ttyUSB0
```

Экспериментальным путем было выяснено, что утилита последовательного терминала minicom [11] способна перехватывать перенаправленные команды и выполнять их на принимающем устройстве (Рис. 1).

```
pi@stand-rpi:~$ echo "uname -a" > /dev/ttyUSB1 #
pi@stand-rpi:~$ # uname -a
Linux mcom02 4.4.189.2 #1 SMP Thu May 19 16:06:19 MSK 2022 armv7l GNU/Linux
#
```

Рис. 1. Перехват команды minicom-ом

2.3.2. Чтение результата

minicom предоставляет возможность обмениваться файлами с последовательным устройством через протоколы xmodem, umodem и zmodem [12]. Однако пользователю требуется прожимать комбинации клавиш, чтобы запустить передачу файла, найти сам файл и выбрать адрес копирования. Данные комбинации не считываются при перенаправлении командой echo. Для автоматизированной диагностики данный способ не подходит.

Тогда был найден способ мониторить все, что происходит во время выполнения minicom. Команда cat /dev/ttyUSB выводит все, что печатается в последовательный порт. Однако экспериментальным путем было выявлено, что некоторые строки не успевают считываться, тем самым пропадала часть информации. Для решения этой проблемы был найден аналог команды cat - strace.

Команда strace выводит все вызовы функций, которые вызывает процесс в linux. Для работы была настроена команда strace на процесс minicom, с заданием фильтра на вызов функции read():

```
strace -e trace=read -s 150 -p <minicom pid> -o output.txt
```

После завершения выполнения программы создается текстовый документ output.txt, содержащий в себе весь журнал внутренних событий. Содержимое этого файла представлено на рисунке 2.



```
Файл Правка Формат Вид Справка
read(0, "\33OA", 32) = 3
read(3, "\r# python hello.py \33[J", 127) = 22
read(0, "\r", 32) = 1
read(3, "\r\n", 127) = 2
read(3, "-- Logs begin at Fri 2023-07-14 09:35:49 UTC, end at Tue 2023-07-18 13:50:11 UTC. --\r\nJul 18 07:46:31 mcom02 kernel: Booting Li", 127) = 127
read(3, "nux on physical CPU 0x0\r\n", 127) = 25
read(3, "Jul 18 07:46:31 mcom02 kernel: Linux version 4.4.189.2 (sprainbrains@DELL-Laptop-VKIST) (gcc version 7.3.0 (Buildroot 3.1.0-201", 127) = 127
read(3, "9-09-27) ) #1 SMP Thu May 19 16:06:19 MSK 2022\r\nJul 18 07:46:31 mcom02 kernel: CPU: ARMv7 Processor [413fc090] revision 0 (ARMv", 127) = 127
read(3, "7)", 127) = 2
read(3, ", cr=10c5387d\r\nJul 18 07:46:31 mcom02 kernel: CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache\r\nJul 18", 127) = 127
read(3, "07:46:31 mcom02 kernel: Machine model: Salute-EL24PM2 r1.0-1.1, Salute-EL24OM1 r1.2\r\nJul 18 07:46:31 mcom02 kernel: cma: Reserv", 127) = 127
read(3, "ed 128 MiB at 0xd8000000\r\n", 127) = 26
read(3, "Jul 18 07:46:31 mcom02 kernel: Memory policy: Data cache wwritealloc\r\n", 127) = 69
read(3, "Jul 18 07:46:31 mcom02 kernel: On node 0 totalpages: 524288\r\n", 127) = 61
read(3, "Jul 18 07:46:31 mcom02 kernel: free_area_init_node: node 0, pgdat c074ec00, node_mem_map eebf8000\r\n", 127) = 99
read(3, "Jul 18 07:46:31 mcom02 kernel: Normal zone: 1536 pages used for memmap\r\n", 127) = 74
read(3, "Jul 18 07:46:31 mcom02 kernel: Normal zone: 0 pages reserved\r\n", 127) = 64
read(3, "Jul 18 07:46:31 mcom02 kernel: Normal zone: 196608 pages, LIFO batch:31\r\n", 127) = 75
read(3, "Jul 18 07:46:31 mcom02 kernel: HighMem zone: 327680 pages, LIFO batch:31\r\n", 127) = 76
read(3, "Jul 18 07:46:31 mcom02 kernel: PERCPU: Embedded 12 pages/cpu @eebc1000 s17548 r8192 d23412 u49152\r\n", 127) = 99
read(3, "Jul 18 07:46:31 mcom02 kernel: pcpu-alloc: s17548 r8192 d23412 u49152 alloc=12*4096\r\n", 127) = 85
read(3, "Jul 18 07:46:31 mcom02 kernel: pcpu-alloc: [0] 0 [0] 1 \r\n", 127) = 57
read(3, "Jul 18 07:46:31 mcom02 kernel: Built 1 zonelists in Zone order, mobility grouping on. Total pages: 522752\r\n", 127) = 108
read(3, "Jul 18 07:46:31 mcom02 kernel: Kernel command line: console=ttyS0,115200 root=/dev/mmcblk1p2 rootfstype=ext4 rw rootwait video=", 127) = 127
read(3, "HDMI:1920x1080\r\nJul 18 07:46:31 mcom02 kernel: PID hash table entries: 4096 (order: 2, 16384 bytes)\r\n", 127) = 101
read(3, "Jul 18 07:46:31 mcom02 kernel: Dentry cache hash table entries: 131072 (order: 7, 524288 bytes)\r\n", 127) = 97
read(3, "Jul 18 07:46:31 mcom02 kernel: Inode-cache hash table entries: 65536 (order: 6, 262144 bytes)\r\n", 127) = 95
read(3, "Jul 18 07:46:31 mcom02 kernel: Memory: 1940752K/2097152K available (5399K kernel code, 174K rwdata, 1596K rodata, 284K init, 37", 127) = 127
read(3, "8K bss, 25328K reserved, 131072K cma-reserved, 1179648K highmem)\r\nJul 18 07:46:31 mcom02 kernel: Virtual kernel memory layout:\r", 127) = 127
read(3, "\n", 127) = 2
read(3, "
0xffff0000 - 0xffff1000 ( 4 kB)\r\n", 127) = 81
read(3, "
fixmap : 0xffc00000 - 0xffd00000 (3072 kB)\r\n", 127) = 82
read(3, "
vmalloc : 0xf0800000 - 0xff800000 ( 240 MB)\r\n", 127) = 82
read(3, "
lowmem : 0xc0000000 - 0xf0000000 ( 768 MB)\r\n", 127) = 82
read(3, "
pkmap : 0xbfe00000 - 0xc0000000 ( 2 MB)\r\n", 127) = 82
read(3, "
modules : 0xbf000000 - 0xbfe00000 ( 14 MB)\r\n", 127) = 82
read(3, "
.text : 0xc0008000 - 0xc06dcf5c (6996 kB)\r\n", 127) = 82
read(3, "
.init : 0xc06dd000 - 0xc0724000 ( 284 kB)\r\n", 127) = 82
read(3, "
.data : 0xc0724000 - 0xc074f000 ( 175 kB)\r\n", 127) = 82
read(3, "
.bss : 0xc0752000 - 0xc07b0914 ( 379 kB)\r\n", 127) = 82
read(3, "Jul 18 07:46:31 mcom02 kernel: SLUB: HWalgn=64, Order=0-3, MinObjects=0, CPUs=2, Nodes=1\r\n", 127) = 91
read(3, "Jul 18 07:46:31 mcom02 kernel: Hierarchical RCU implementation.\r\n", 127) = 65
read(3, "Jul 18 07:46:31 mcom02 kernel: Build-time adjustment of leaf fanout to 32.\r\n", 127) = 84
read(3, "Jul 18 07:46:31 mcom02 kernel: NR_IRQS:16 nr_irqs:16 16\r\n", 127) = 57
read(3, "Jul 18 07:46:31 mcom02 kernel: L2C: platform modifies aux control register: 0x02070000 -> 0x02470000\r\n", 127) = 102
read(3, "Jul 18 07:46:31 mcom02 kernel: L2C: DT/platform modifies aux control register: 0x02070000 -> 0x02470000\r\n", 127) = 105
read(3, "Jul 18 07:46:31 mcom02 kernel: L2C-310 enabling early BRESP for Cortex-A9\r\n", 127) = 75
read(3, "Jul 18 07:46:31 mcom02 kernel: L2C-310 full line of zeros enabled for Cortex-A9\r\n", 127) = 81
read(3, "Jul 18 07:46:31 mcom02 kernel: L2C-310 dynamic clock gating enabled, standby mode enabled\r\n", 127) = 91
read(3, "Jul 18 07:46:31 mcom02 kernel: L2C-310 cache controller enabled, 16 ways, 1024 kB\r\n", 127) = 83
read(3, "Jul 18 07:46:31 mcom02 kernel: L2C-310: CACHE_ID 0x410000c9, AUX_CTRL 0x46470001\r\n", 127) = 82
read(3, "Jul 18 07:46:31 mcom02 kernel: clocksource: timer: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 13272641988 ns\r\nJul 18", 127) = 127
read(3, "07:46:31 mcom02 kernel: sched_clock: 32 bits at 144MHz, resolution 6ns, wraps every 14913080828ns\r\n", 127) = 100
read(3, "Jul 18 07:46:31 mcom02 kernel: Console: colour dummy device 80x30\r\n", 127) = 67
read(3, "Jul 18 07:46:31 mcom02 kernel: Calibratin delay loop... 1626.93 BogoMIPS (lni=8134656)\r\n", 127) = 89
```

Windows (CRLF)

Стр 1, из 1

Рис. 2. Результат работы strace

Преимущества способа:

- Для диагностики устройства достаточно запустить программное обеспечение на управляющем компьютере. Нет необходимости ставить дополнительные скрипты и обработчики на диагностируемые устройства.

Недостатки способа:

- Без дополнительного окна терминала, в котором будет запущен minicom, способ работать не будет.
- Необходимо запустить команду strace в отдельном терминале, так как через основной терминал еще необходимо останавливать выполнение minicom и strace.

2.4. minicom и Qt C++

Для решения недостатков из предыдущего пункта, было принято использовать фреймворк Qt с его механизмами многопоточности, так как при этом способе не было необходимости ставить дополнительные обработчики и скрипты на диагностируемые устройства.



Для вызова функции в новом потоке необходимо создать объект класса `QFuture` и присвоить ему значение, которая возвращает функция `QtConcurrent::run()`:

```
QFuture<void> name_thread = QtConcurrent::run(thread_function);
```

В главном потоке программы отсылаются команды через функцию `system()` с перенаправлением в файл `ttyUSB`:

```
system("echo `uname -a` > /dev/ttyUSB ");
```

После выполнения программы, на управляющем компьютере создается файл с журналом всех внутренних событий диагностируемого устройства. После синтаксической обработки текстового файла информация готова к отображению в графической оболочке.

3. Графический интерфейс программного обеспечения для мониторинга состояния устройств в УНУ АССИСТ

3.1. Описание архитектуры

Графическое программное обеспечение (Далее - ПО) имеет несколько классов, которые выполняют свои определенные функции. Опишем их ниже.

`Application Context` - управляющий класс, который принимает в себя все зависимости и является связующим звеном между классами. Через этот класс осуществляется управление всем приложением.

`Execution Logs` - класс, объект которого нельзя создать. Этот класс имеет только статические методы и статический массив, который является журналом событий, включающий в себя все, что происходит по мере выполнения приложения.

`Data Collector` - класс, который собирает всю необходимую информацию. Первое - документация о каждом устройстве стенда УНУ АССИСТ. Второе - это состояние устройств.

Собрав всю информацию из `Data Collector`, `Application Context` направляет данные в класс `Main Window`, в котором они будут отображаться графически.

При этом `Main Window` использует в себе функционал класса `Scheme Label Widget`. Это пользовательский виджет, который включает в себя изображение, реагирует на нажатие на себя и выдает сигнал об этом событии.

Для удобного просмотра передаваемого пакета `SpaceWire` создан класс `Package Panel`. Это пользовательское диалоговое окно, содержащее свои виджеты, которые, в свою очередь, отображает один из всех переданных пакетов.

`Journals Panel` - класс - пользовательское диалоговое окно - который предоставляет доступ к имеющимся журналам последовательных устройств и возможность генерировать журналы.

Схема архитектуры программного обеспечения продемонстрирована на рисунке 3.

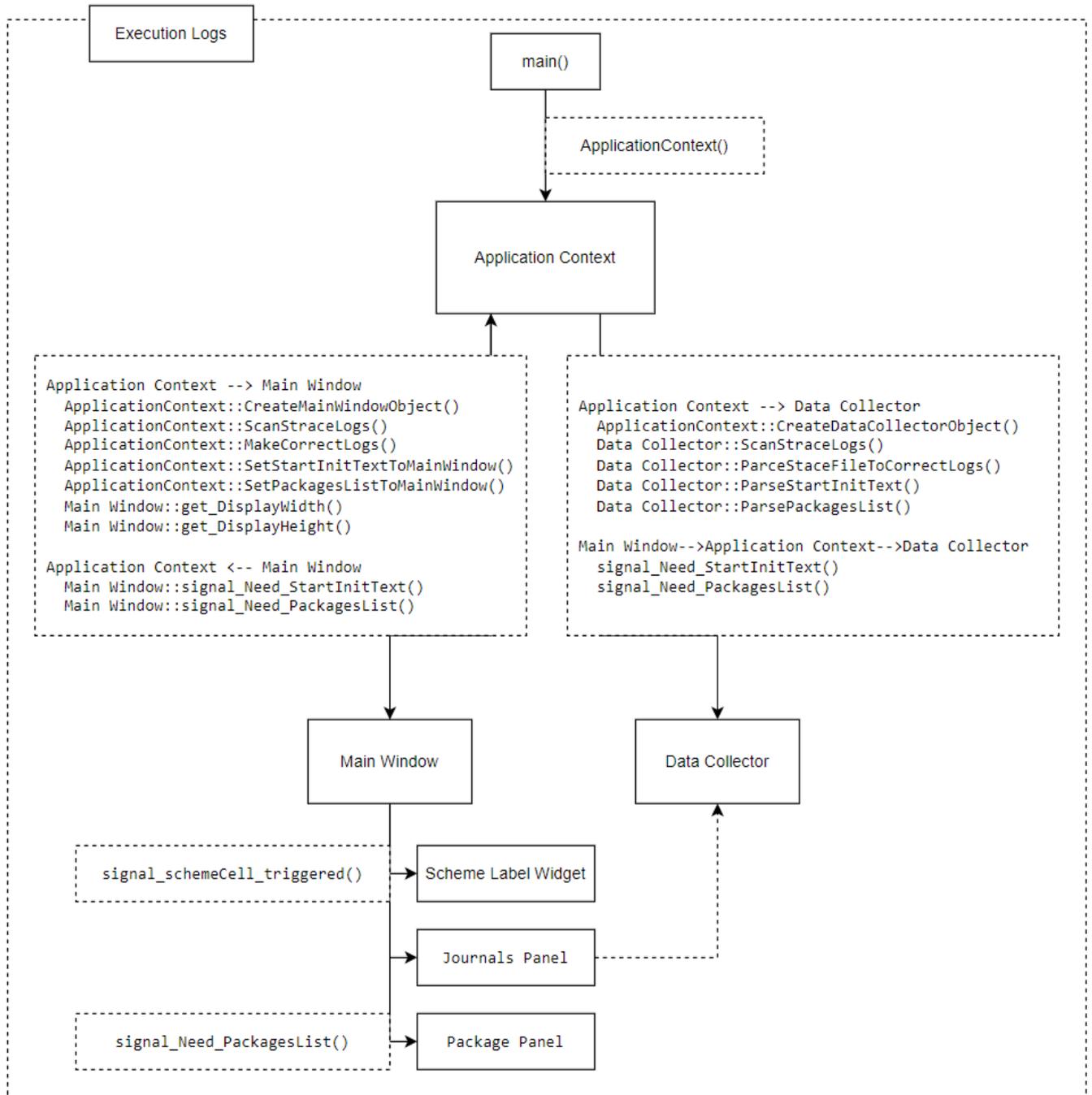


Рис. 3. Схема архитектуры ПО

3.2. Демонстрация работы

При запуске приложения открывается интерактивная схема стенда, на которой можно выбрать устройство. По умолчанию устройство не выбрано (Рис. 4).

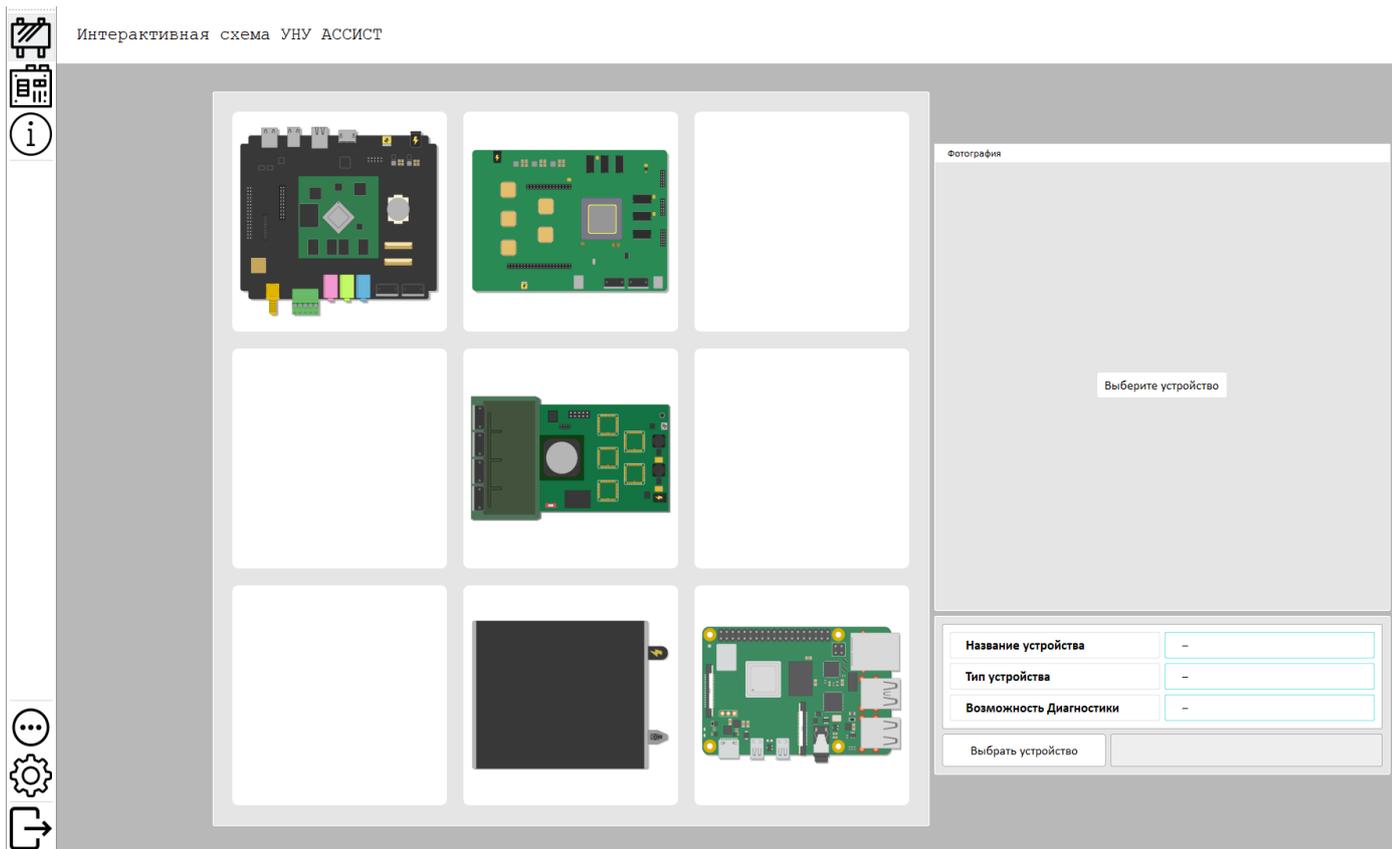


Рис. 4. Запуск приложения

В левой части экрана расположена панель кнопок:

- Интерактивная схема
- Состояние устройства
- Информация о приложении
- Информация о УНУ АССИСТ
- Настройки
- Завершение работы приложения

С помощью этой панели можно переключаться между вкладками приложения или выполнять определенные действия.

Если на интерактивной схеме устройство не выбрано, то при переходе на вкладку “Состояние устройства” появится предупреждение, показанное на рисунке 5, и переход не будет выполнен.

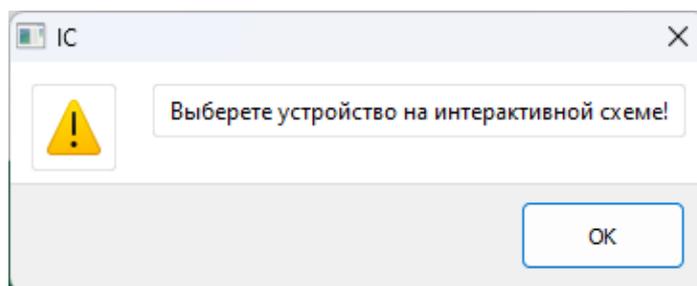


Рис. 5. Переход во вкладку состояния устройств



При нажатии на изображение схемы устройства отображается более крупное изображение устройства и краткая информация о нем (Рис. 6).

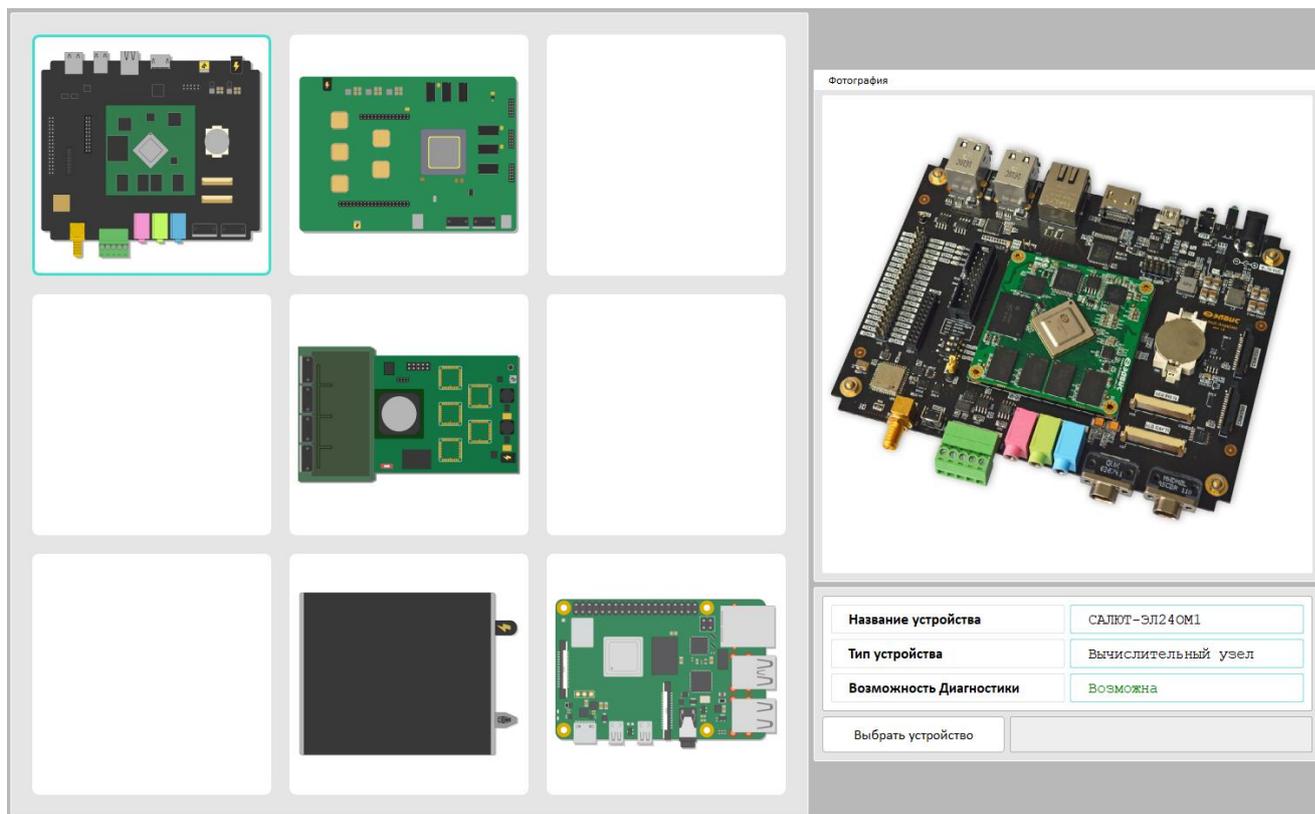


Рис. 6. На схеме выбрано устройство

При нажатии на кнопку “Выбрать устройство” пользователь перейдет во вкладку “Состояние устройства”. В левой части экрана появится панель, содержащая фотографию устройства, его краткое описание и панель кнопок, с помощью которых происходит отображается дополнительная информация (Рис. 7).



Название	САЛЮТ-ЭЛ24ОМ1
Тип	Вычислительный узел
Диагностика	Возможна

Документация

Инициализация запуска

Панель сканирования

Обработка пакетов

Схема

Рис. 7. На схеме выбрано устройство

В данной вкладке в первую очередь перед пользователем покажется появится краткое описание устройства (Рисунок 8), его документация (Рисунок 9) и ссылка на официального производителя (Рисунок 10).

Описание

Модуль отладочный Салют-ЭЛ24ОМ предназначен для изучения аппаратно-программных средств процессорного модуля Салют-ЭЛ24ПМ, отладки прикладных программ пользователя, макетирования систем интеллектуального управления, цифровой обработки сигналов, ввода/вывода и обработки видео.

Рис. 8. Краткое описание устройства

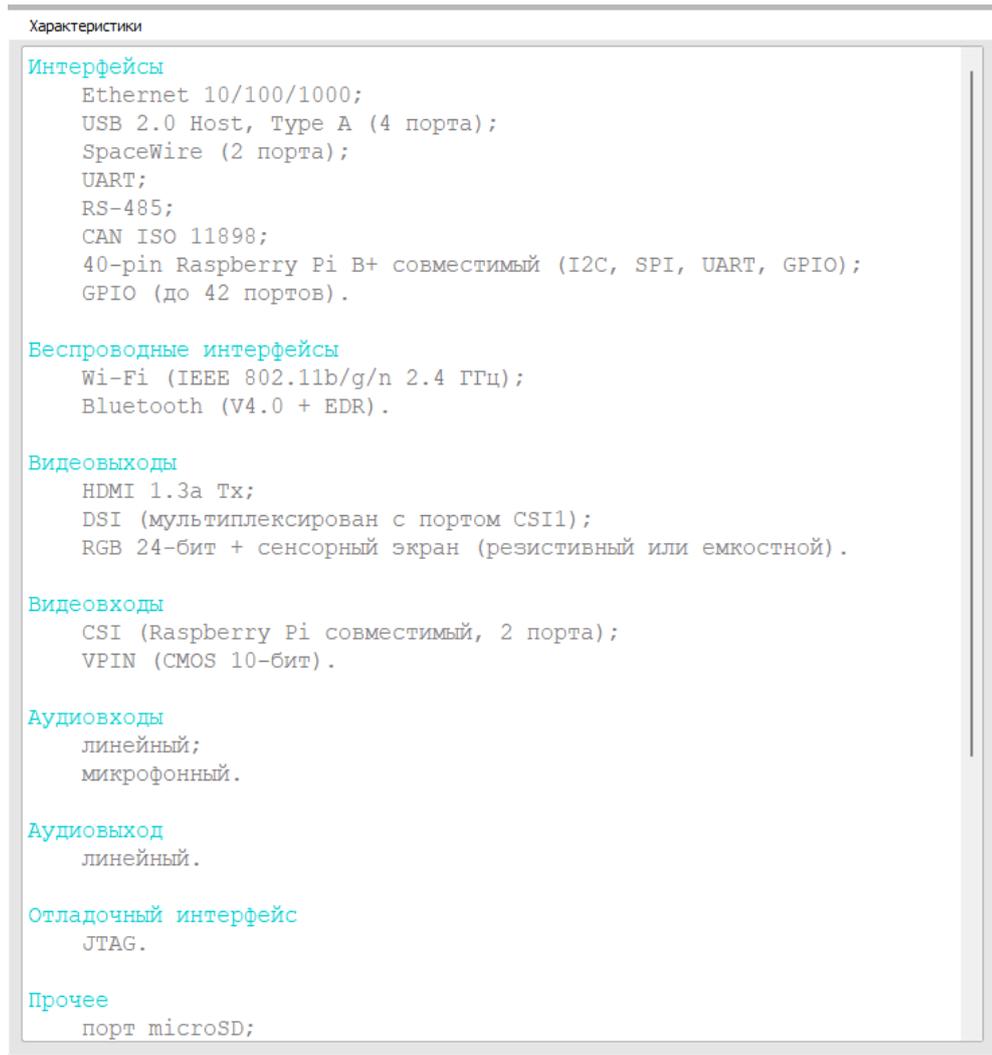


Рис. 9. Документация устройства

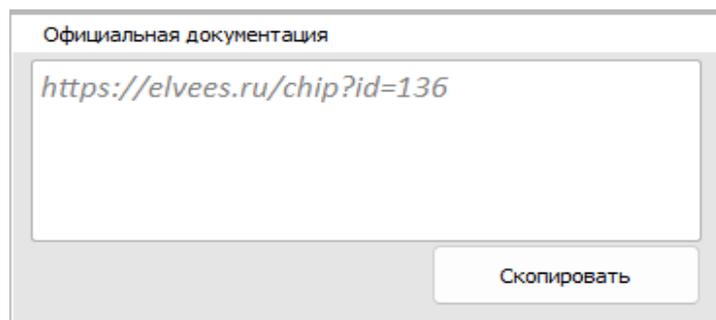


Рис.10. Ссылка на производителя

При нажатии на кнопку “Инициализация Запуска” отобразится информация, которая генерируется устройством при запуске (Рис. 11). В поле “Инициализация при запуске устройства” выведена информация о регистрах и стартовой инициализации устройства. В поле “Состояние каналов SpaceWire” отображается состояние всех каналов SpaceWire (Рис. 12).



```
Инициализация при запуске устройства

_journal_check_start:56

> START INIT :
Jul 18 08:00:17 mcom02 kernel: swic_init:1798
Jul 18 08:00:17 mcom02 kernel: swic_stats:1221 swc0
Jul 18 08:00:17 mcom02 kernel: base_swic[0] F12FC000.
Jul 18 08:00:17 mcom02 kernel: swic_link_irq(i) = 41
Jul 18 08:00:17 mcom02 kernel: swic_err_irq(i) = 40
Jul 18 08:00:17 mcom02 kernel: swic_stats:1221 swc0
Jul 18 08:00:17 mcom02 kernel: swic_time_irq(i) = 39
Jul 18 08:00:17 mcom02 kernel: swic_rx_desc_irq(i) = 38
Jul 18 08:00:17 mcom02 kernel: swic_tx_data_irq(i) = 40

> START INIT :
Jul 18 08:00:17 mcom02 kernel: swic_init:1798
Jul 18 08:00:17 mcom02 kernel: swic_stats:1221 swc1
Jul 18 08:00:17 mcom02 kernel: base_swic[1] F1312000.
Jul 18 08:00:17 mcom02 kernel: swic_link_irq(i) = 46
Jul 18 08:00:17 mcom02 kernel: swic_err_irq(i) = 45
Jul 18 08:00:17 mcom02 kernel: swic_time_irq(i) = 44
Jul 18 08:00:17 mcom02 kernel: swic_rx_desc_irq(i) = 43
Jul 18 08:00:17 mcom02 kernel: swic_tx_data_irq(i) = 45
Jul 18 08:00:17 mcom02 kernel:
=====ALL REGISTERS ( SWIC0 )=====
Jul 18 08:00:17 mcom02 kernel: HW_VER 5 (F12FC000)
Jul 18 08:00:17 mcom02 kernel: STATUS 600A00 (F12FC004), DS(state) 0
Jul 18 08:00:17 mcom02 kernel: swic_stats:1221 swc1
Jul 18 08:00:17 mcom02 kernel: RX_CODE 0 (F12FC008)
Jul 18 08:00:17 mcom02 kernel: MODE_CR C4004 (F12FC00C)
Jul 18 08:00:17 mcom02 kernel: TX_SPEED FC0300 (F12FC010)
Jul 18 08:00:17 mcom02 kernel: TX_CODE 0 (F12FC014)
Jul 18 08:00:17 mcom02 kernel: RX_SPEED 43 (F12FC018)
Jul 18 08:00:17 mcom02 kernel: CNT_RX_PACK 0 (F12FC020)
Jul 18 08:00:17 mcom02 kernel: CNT_RX0_PACK 0 (F12FC01C)
Jul 18 08:00:17 mcom02 kernel: ISR_L 0 (F12FC024)
Jul 18 08:00:17 mcom02 kernel: ISR_H 0 (F12FC028)
Jul 18 08:00:17 mcom02 kernel: TRUE_TIME 0 (F12FC02C)
Jul 18 08:00:17 mcom02 kernel: TOUT_CODE 0 (F12FC030)
Jul 18 08:00:17 mcom02 kernel: ISR_tout_L 0 (F12FC034)
Jul 18 08:00:17 mcom02 kernel: ISR_tout_H 0 (F12FC038)
Jul 18 08:00:17 mcom02 kernel: LOG_ADDR 0 (F12FC03C)
Jul 18 08:00:17 mcom02 kernel: =====
Jul 18 08:00:17 mcom02 kernel:
=====ALL REGISTERS ( SWIC1 )=====
Jul 18 08:00:17 mcom02 kernel: HW_VER 5 (F1312000)
Jul 18 08:00:17 mcom02 kernel: STATUS 8A20 (F1312004), DS(state) 1
Jul 18 08:00:17 mcom02 kernel: RX_CODE 0 (F1312008)
Jul 18 08:00:17 mcom02 kernel: MODE_CR C4004 (F131200C)
Jul 18 08:00:17 mcom02 kernel: TX_SPEED FC0300 (F1312010)
Jul 18 08:00:17 mcom02 kernel: TX_CODE 0 (F1312014)
Jul 18 08:00:17 mcom02 kernel: RX_SPEED 0 (F1312018)
Jul 18 08:00:17 mcom02 kernel: CNT_RX_PACK 0 (F131201C)
```

Рис. 11. Инициализация запуска

```
Состояние каналов SpaceWire

> chnstate_spw :> chnstate_spw :
Jul 18 08:00:18 mcom02 kernel: swc0 (port 0) swic_ioctl:1491
Jul 18 08:00:18 mcom02 kernel: cmd 89F0
< END chnstate_spw

> chnstate_spw :> chnstate_spw :
Jul 18 08:00:18 mcom02 kernel: swc1 (port 1) swic_ioctl:1491
Jul 18 08:00:18 mcom02 kernel: cmd 89F0
< END chnstate_spw
```

Рис. 12. Состояние каналов SpaceWire



При нажатии на кнопку “Панель Журналов” покажется диалоговое окно, которое будет включать в себя виджеты, представляющие управление текстовыми документами с журналами событий выбранного устройства (Рис. 13).

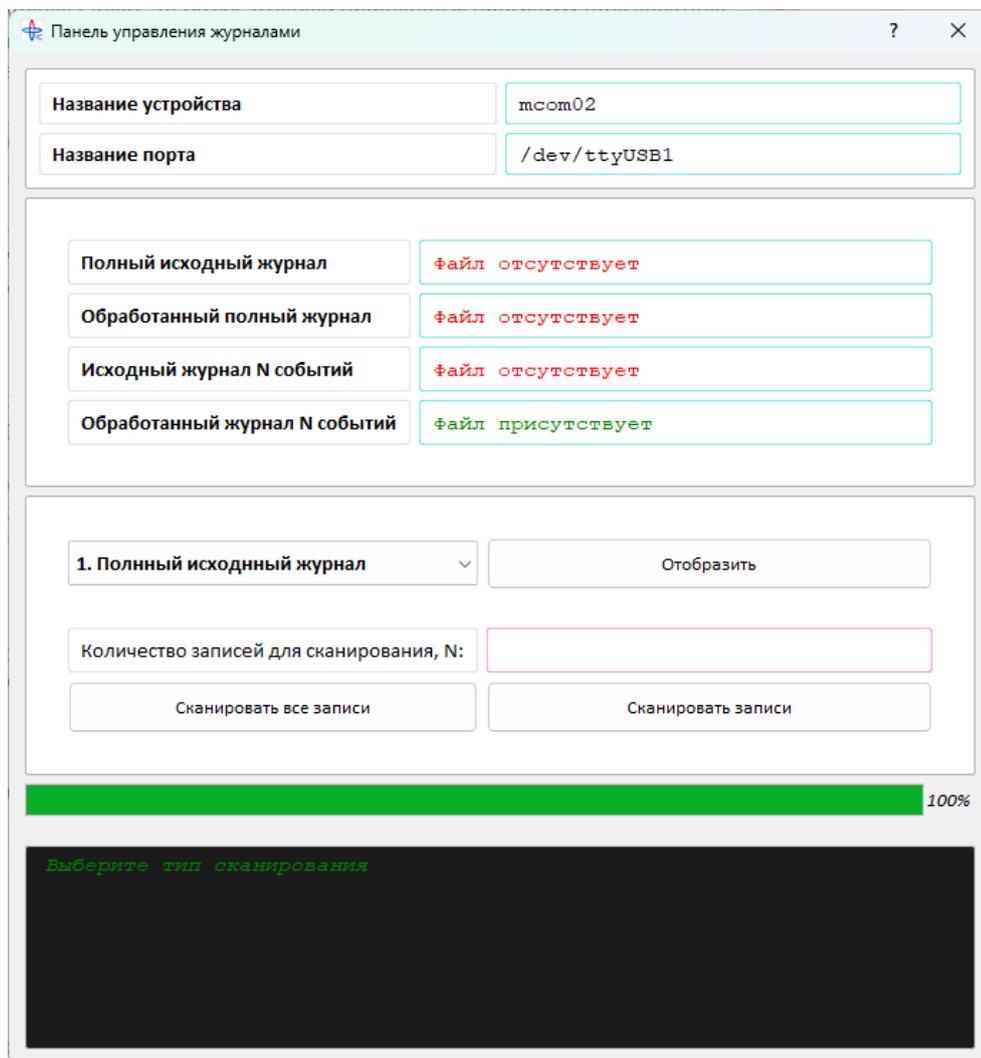


Рис. 13. Панель журналов

При выборе не диагностируемого устройства кнопки “Инициализация запуска”, “Панель Журналов” и “Обработка пакетов” выдают предупреждение о том, что устройство невозможно диагностировать (Рис. 14).

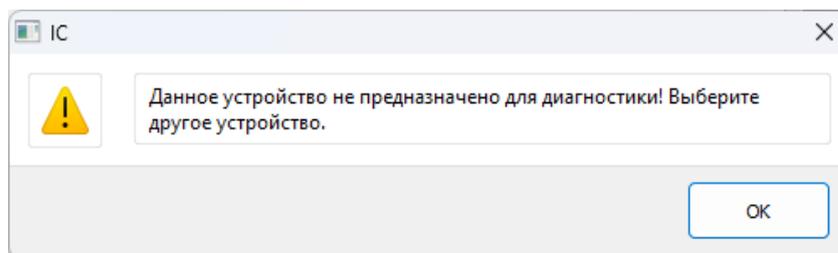


Рис. 14. Предупреждение крупным планом

При переходе на вкладку “Информация о приложении” перед пользователем покажется основная информация о приложении (Рис. 15).

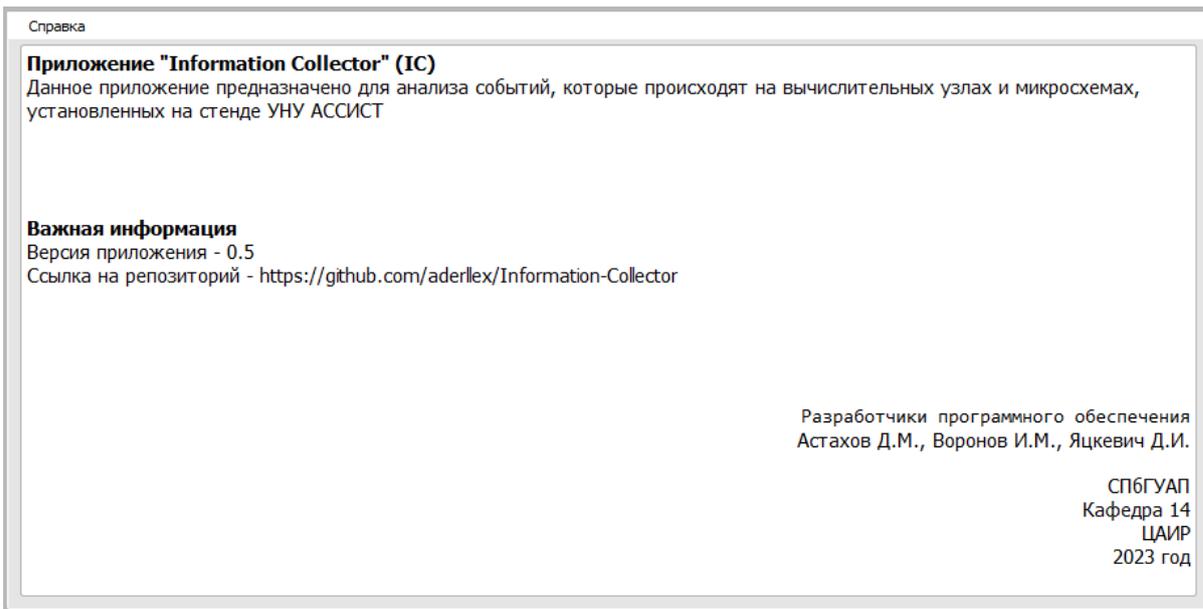


Рис. 17. Краткая информация

Заключение

В данной статье была представлена информация о разработке графического программного обеспечения, которое предназначалось для диагностики внешнего устройства через UART с использованием терминала последовательного соединения minicom. Весь процесс диагностики проводится программным обеспечением, находящимся только на управляющем компьютере. Нет необходимости ставить дополнительное ПО на диагностируемое устройство. В связи с этим данный способ диагностики является особенно актуальным среди остальных.

В результате были проанализированы механизмы диагностики внешнего последовательного устройства, был разработан уникальный алгоритм сбора журналов и было продемонстрировано графическое программное обеспечение для отображения информации внутренних событий.

Ближайшая разработка включает в себя внедрение механизма сбора информации в графическую оболочку, управление которым будет происходить из диалогового окна "Панель Журналов", добавление информации во вкладку "Информация о УНУ АССИСТ", указанной на официальном сайте [13], и создание системы настроек приложения, которые будут включать в себя настройки языка, цветовой темы и параметры запуска приложения.

В итоге можно сказать, что поставленная цель по разработке графического программного обеспечения для мониторинга состояния последовательных устройств частично выполнена. На данный момент ведется работа с настройкой ПО под MC-30SF6EM-6U, а также реализацией универсального интерфейса для сторонних разработчиков, которые будут приносить свои устройства для тестирования на УНУ АССИСТ.

Данная работа выполнена при финансовой поддержке Министерства науки и высшего образования Российской Федерации, соглашение № FSRF-2023-0003, "Фундаментальные основы построения помехозащищенных систем космической и спутниковой связи, относительной навигации, технического зрения и аэрокосмического мониторинга".

СПИСОК ЛИТЕРАТУРЫ

1. Синёв Н. И. Аппаратно-программное исследование коммуникационных протоколов для бортовых сетей / Н. И. Синёв, А. А. Карандашев, В. Л. Оленев, Н. Ю. Чумакова, А. Ю. Сыщиков // Санкт-Петербургский государственный университет аэрокосмического приборостроения. – 2022. – С. 18.



2. Салют-ЭЛ24ОМ1 [Электронный ресурс]. – URL: <https://elvees.ru/chip?id=136/> (дата обращения: 13.09.2023)
3. МС-30SF6ЕМ-6U [Электронный ресурс]. – URL: <https://elvees.ru/chip?id=1405/> (дата обращения: 13.09.2023)
4. МСК-02РЕМ-3U [Электронный ресурс]. – URL: <https://elvees.ru/chip?id=1372/> (дата обращения: 13.09.2023)
5. Raspberry Pi [Электронный ресурс]. – URL: <https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/> (дата обращения: 13.09.2023)
6. SpaceWire Standard. Space Technology Centre. University of Dundee ECSS – Space Engineering. "SpaceWire – Links, Nodes, Routers and Networks". ECSS-E-ST-50-12 Draft H1. – 2018. – 124 p.
7. Raspberry Pi OS [Электронный ресурс]. – URL: <https://www.raspberrypi.com/software/> (Дата обращения: 13.09.2023)
8. Qt 5 документация [Электронный ресурс]. – URL: <https://www.qt.io/> (дата обращения: 13.09.2023)
9. Habr: Работа с СОМ-портом на Си в linux [Электронный ресурс]. – URL: <https://habr.com/ru/companies/ruvds/articles/578432/> (дата обращения: 13.09.2023)
10. QSerialPort. Qt 5.15 Documentation [Электронный ресурс]. – URL: <https://doc.qt.io/qt-5/qserialport.html> (дата обращения: 13.09.2023)
11. Ubuntu Manpage: minicom - friendly serial communication program [Электронный ресурс]. – URL: <https://manpages.ubuntu.com/manpages/trusty/man1/minicom.1.html> (дата обращения: 13.09.2023)
12. Ubuntu Manpage: rx, rb, rz - XMODEM, YMODEM, ZMODEM (Batch) file receive [Электронный ресурс]. – URL: <https://manpages.ubuntu.com/manpages/xenial/man1/rz.1.html> (дата обращения: 13.09.2023)
13. ГУАП: ЦАИР – УНУ АССИСТ [Электронный ресурс]. – URL: <https://guap.ru/m/assist-spw/> (дата обращения: 13.09.2023)

ИНФОРМАЦИЯ ОБ АВТОРАХ

Синёв Николай Иванович –

Ст. преподаватель, мл. науч. сотрудник

Санкт-Петербургский государственный университет аэрокосмического приборостроения
190000, Санкт-Петербург, ул. Большая Морская, д. 67, лит. А

E-mail: nikolay.sinyov@guap.ru

Астахов Даниил Максимович –

Студент

Санкт-Петербургский государственный университет аэрокосмического приборостроения
190000, Санкт-Петербург, ул. Большая Морская, д. 67, лит. А

E-mail: daniil-actaxov@yandex.ru

Воронов Илья Михайлович –

Студент кафедры аэрокосмических компьютерных программных систем

Санкт-Петербургский государственный университет аэрокосмического приборостроения
190000, Санкт-Петербург, ул. Большая Морская, д. 67, лит. А

E-mail: ilya.stud.spb@gmail.com

Яцкевич Даниил Игоревич –

Студент

Санкт-Петербургский государственный университет аэрокосмического приборостроения
190000, Санкт-Петербург, ул. Большая Морская, д. 67, лит. А

E-mail: j.daniil@bk.ru



INFORMATION ABOUT THE AUTHORS

Sinev Nikolay Ivanovich –

Senior Lecturer, Associate Junior Research
Saint-Petersburg State University of Aerospace Instrumentation
67, Bolshaya Morskaia str., Saint-Petersburg, 190000, Russia
E-mail: nikolay.sinyov@guap.ru

Astakhov Daniil Maksimovich –

Student
Saint-Petersburg State University of Aerospace Instrumentation
67, Bolshaya Morskaia str., Saint-Petersburg, 190000, Russia
E-mail: daniil-actaxov@yandex.ru

Voronov Ilya Mikhailovich –

Student of the Departments of Aerospace Computer Software Systems
Saint-Petersburg State University of Aerospace Instrumentation
67, Bolshaya Morskaia str., Saint-Petersburg, 190000, Russia
E-mail: ilya.stud.spb@gmail.com

Yatskevich Daniil Igorevich –

Student
Saint-Petersburg State University of Aerospace Instrumentation
67, Bolshaya Morskaia str., Saint-Petersburg, 190000, Russia
E-mail: j.daniil@bk.ru