Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б11-мм

Расширение функциональности ОСРВ Embox для применения в АСУ ТП

Пилюк Дмитрий Алексеевич

Отчёт по производственной практике
в форме «Производственное задание»

Научный руководитель:
доцент кафедры системного программирования, к. ф.-м. н., Д. В. Луцив

Консультант:
ген. директор ООО «Ембокс» А. В. Бондарев

Санкт-Петербург
2024

Оглавление

Введение

В наше время наблюдается развитие и повсеместное применение технологий автоматизации. Автоматизация позволяет значительно повысить производительность, снизить затраты, минимизировать влияние человеческого фактора и обеспечить выполнение сложных задач с высокой точностью. Внедрение автоматизированных систем управления стало ключевым драйвером прогресса в таких отраслях, как производство, транспорт, энергетика и медицина.

Одной из значимых областей применения автоматизации является управление технологическими процессами. В сложных промышленных системах с большим количеством взаимосвязанных элементов эффективность работы напрямую зависит от скорости обработки информации и точности принимаемых решений. Здесь на первый план выходят автоматизированные системы управления технологическими процессами (АСУ ТП). Они применяются в таких отраслях, как энергетика, нефтегазовая промышленность, машиностроение, химическая и пищевая промышленность.

Для выполнения данных задач широко используются операционные системы реального времени, которые способны гарантировать строгое соблюдение временных ограничений. Одной из таких ОСРВ является Embox, перспективное решение в данной области благодаря своим особенностям.

1. Постановка задачи

Целью работы является расширение возможностей ОСРВ Embox для использования Embox в сфере АСУ ТП. Для её выполнения были поставлены следующие задачи:

1. Изучить существующие открытые решения в области АСУ ТП.
2. Предоставить среду разработки, поддерживающую создание и редактирование кода языков стандарта IEC 61131-3 для Embox.
3. Добавить в Embox поддержку необходимых модулей.
4. Поставить эксперимент.

2. Обзор

2.1. ОСРВ Embox

Embox [2] — это конфигурируемая операционная система реального времени, предназначенная для встраиваемых систем с ограниченными ресурсами. Разработка начиналась в 2010 году на кафедре системного программирования Математико-механического факультета СПбГУ как студенческий проект.

Исходный код ОСРВ Embox представляет собой модульную систему, что позволяет гибко настраивать конечную сборку под конкретную задачу, с возможностью включить только минимально необходимые компоненты, либо получить ОС с богатой функциональностью. Эта модульность достигается за счёт использования системы сборки Mybuild.

Embox предоставляет POSIX-совместимость, благодаря чему позволяет портировать многие Linux приложения. Так же в состав проекта входит стандартная библиотека языка C.

Embox совмещает в себе возможность выполнять сложные функциональные задачи и является системой реального времени. Кроме того ОС имеет развитый сетевой стек, набор различных приложений, библиотек и при этом низкие аппаратные требования, поэтому хорошо подходит для использования в сфере АСУ ТП.

2.2. АСУ ТП

Автоматизированная система управления технологическим процессом (АСУ ТП) — система технических и программных средств, предназначенных для автоматизации управления технологическим процессом. В качестве аппаратной платформы используются специализированные ПЛК.

Программируемый логический контроллер (ПЛК) — цифровая электронная система, предназначенная для применения в производственной среде, которая использует программируемую память для внутреннего хранения ориентированных на потребителя инструкций по реализации

таких специальных функций, как логика, установление последовательности, согласование по времени, счёт и арифметические действия для контроля посредством цифрового или аналогового ввода/вывода данных различных видов машин или процессов.

Основной задачей в данной области является программирование ПЛК и организация их взаимодействия между собой, что описано в стандарте IEC 61131.

IEC 61131 — набор стандартов МЭК для программируемых контроллеров.

IEC 61131-3 [4] — раздел международного стандарта IEC 61131 , описывающий языки программирования для программируемых логических контроллеров. Включает языки LD, FBD, SFC, ST, IL.

Так же в сфере АСУ ТП применяются различные стандарты взаимодействия, такие как CANopen, OPC-UA, Modbus, EtherCAT

2.3. Среда разработки

Среди открытых решений поддерживающих языки стандарта IEC 61131-3 можно выделить:

- Beremiz — это интегрированная среда разработки для автоматизации машин. Это свободное программное обеспечение, соответствующее, среди прочих стандартов, стандарту IEC-61131. Так же предоставляет свою среду выполнения для программ. Поддерживает различные протоколы взаимодействия. Для генерации программ использует Matiec.

- OpenPLC Editor — форк от Beremiz. Так же предоставляет свою среду выполнения программ.

2.4. Beremiz

Beremiz [1] — это интегрированная среда разработки для автоматизации машин. Это свободное программное обеспечение, соответствующее,

среди прочего, стандарту IEC-61131. Основная идея — использование любого процессора в качестве ПЛК.

Beremiz предоставляет:

- Интегрированная среда разработки (IDE). Графический интерфейс для настройки, написания, сборки и отладки программ ПЛК и управления временем работы ПЛК.
- Интерфейс командной строки (CLI). Создание ПЛК и управление временем работы ПЛК в терминале или с помощью сценария.
- Среда выполнения на Python, запущенная на целевой платформе, взаимодействует с системой ввода-вывода и выполняют программу ПЛК.

В результате сборки проекта генерирует динамическую библиотеку, что не подходит для использования в Embox, поэтому для данной работы главное, что предоставляет Beremiz, это редактор кода и генерация ST кода по остальным языкам стандарта IEC-61131-3.

2.5. Matiec

Matiec [3] — компилятор с открытым исходным кодом для языков программирования, определенных в стандарте IEC 61131-3. Проект предоставляет 2 исполняемых файла iec2c и iec2iec. В данной работе используется iec2c, программа транслирующая код на языке ST в код на языке C.

Компилятор работает в 4 этапа:

- Этап 1 - Лексический анализатор - реализован с помощью flex
- Этап 2 - Синтаксический анализатор - реализован с помощью bison
- Предварительный этап 3 - Заполнение таблиц символов - таблиц символов, которые упростят поиск символов в абстрактном дереве символов.

- Этап 3 - Анализатор семантики - в настоящее время выполняется только проверка типов
- Этап 4 - Генератор кода - генерирует код на ANSI C

В результате работы генерирует следующие файлы:

- config.h
- config.c
- resource1.h
- resource1.c
- POUS.h
- POUS.c
- LOCATED_VARIABLES.h
- VARIABLES.csv

Стоит отметить, что файлы не содержат точки входа в программу, поэтому для запуска необходима сторонняя среда выполнения.

Переменные в программах бывают двух видов: локальные и с адресом. Для тех переменных, которым указан адрес, создаются define, которые так же необходимо реализовать отдельно.

2.6. Modbus

Modbus — это коммуникационный протокол, разработанный в 1979 году компанией Modicon (в настоящее время часть Schneider Electric) для использования в программируемых логических контроллерах. На сегодняшний день Modbus является одним из наиболее широко используемых протоколов в промышленной автоматизации и управления технологическими процессами.

Особенности протокола:

- Архитектура ведущий — ведомый. Ведущее устройство инициирует запросы обмена данными, а ведомые отвечают на них.
- Типы связи:
  - Сеть TCP/IP (Modbus TCP).
  - Последовательные интерфейсы RS-232, RS-422, RS-485.
- Открытая архитектура.

Протокол Modbus определяет несколько типов данных, которые могут быть переданы между устройствами:

- Coils (катушки): одноразрядные цифровые значения, используемые для управления выходами.
- Discrete Inputs (дискретные входы): одноразрядные значения, которые можно только читать.
- Holding Registers (регистры хранения): 16-разрядные значения, доступные для чтения и записи.
- Input Registers (входные регистры): 16-разрядные значения, доступные только для чтения.

3. Реализация

В данной работе реализована возможность создания программ в Beremiz, использующих Modbus server и запуска их на ОСРВ Embox. Для этого необходимо, чтобы в результате сборки Beremiz создавал директорию, содержащую Mybuild файл, header файл для передачи конфигураций для Modbus и код программы на языке st. Так же в embox необходимо реализовать среду выполнения для полученного модуля. Выделяются 3 основных раздела:

- Добавление цели сборки для Embox в Beremiz.
- Beremiz Modbus plugin.
- Среда выполнения в Embox.

3.1. Добавление цели сборки для Embox в Beremiz

Был сделан форк репозитория Beremiz[1]. Модифицирован процесс сборки для цели Embox. Beremiz генерирует ST код программы, Mybuild файл с необходимыми зависимостями и header файл, если используется Modbus.

3.2. Modbus Plugin Beremiz

При сборке для цели Embox происходит проверка использования Modbus в beremiz-проекте. Если он подключен, то происходит генерация header файла для инициализации переменных, которые ссылаются на память Modbus сервера, и параметров серверов. Для этого был добавлен плагин modbus_embox в проект Beremiz[2]. Файл mb_runtime.h предоставляет шаблон для генерации header файла, код представлен в листинге 1. Структура server_node_t содержит необходимые поля для запуска Modbus сервера с использованием libmodbus. На текущий

[1]Репозиторий Beremiz https://github.com/beremiz/beremiz, форк репозитория https://github.com/embox/beremiz/tree/add_embox_target

[2]Рабочий репозиторий Beremiz https://github.com/embox/beremiz/tree/add_embox_target

момент реализована поддержка TCP серверов. В конце файла объявляются переменные и макрос для их инициализации.

Листинг 1: beremiz/modbus_embox/mb_runtime.h

```
#include "modbus.h"


typedef struct _server_node_t {
    const char *ip_adress;
    uint16_t port;
    uint8_t slave_id;
    modbus_mapping_t mem_area;
    modbus_t *ctx;
} server_node_t;


/* Values for instance %(locstr)s \
of the modbus plugin */

#define NUMBER_OF_TCPSERVER_NODES %(tcpserver_node_count)s

#define NUMBER_OF_SERVER_NODES NUMBER_OF_TCPSERVER_NODES


/*initialization following all parameters \
given by user in application*/

static server_node_t server_nodes[NUMBER_OF_SERVER_NODES] = {
%(server_nodes_params)s
}
;

/*******************/
```

```
/*located variables*/
/******************/

%(loc_vars)s

#define LOC_VARS_INIT %(loc_vars_init)s
```

Эти данные заполняются из экранных форм приложения 1 и 2. Реализация представлена в репозитории.

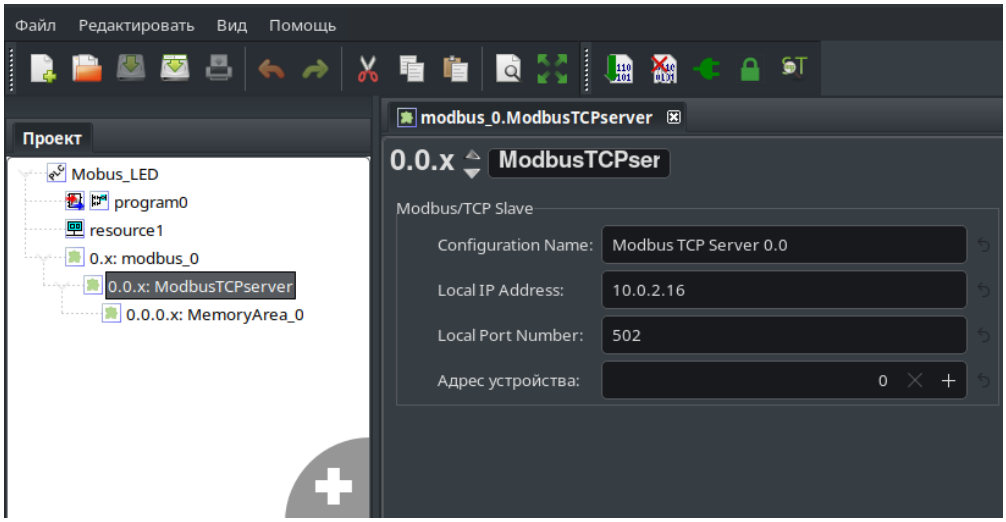

Рис. 1: Создание Modbus сервера в Beremiz.

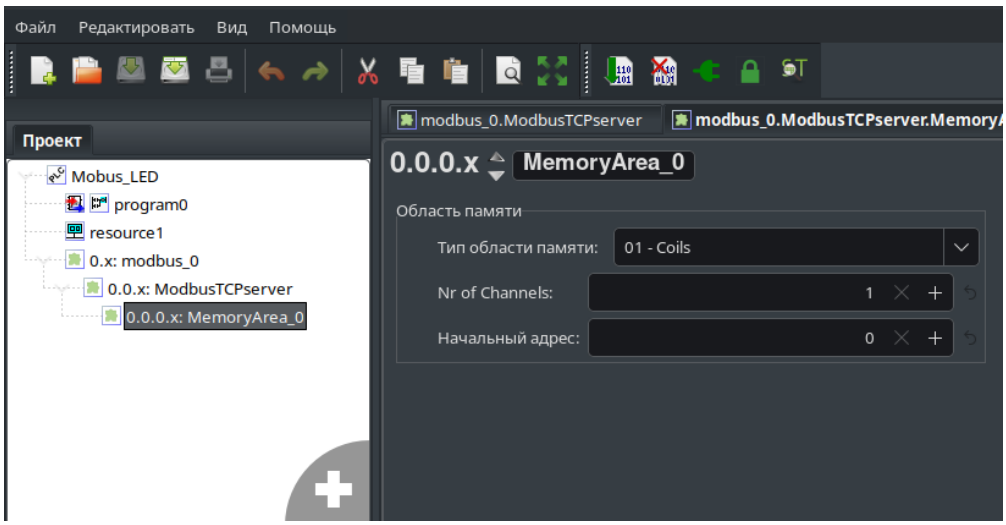

Рис. 2: Добавление области памяти modbus сервера в Beremiz.

3.3. Среда выполнения в Embox

Так как в работе рассматриваются программы, использующие Modbus, необходимо предоставить такую среду выполнения, в которой одновременно будет реализовано выполнение основной программы и работа Modbus сервера. Для этого было использованы потоки (POSIX-threads).

Необходимо реализовать следующие этапы программы:

1. Инициализация каждого Modbus сервера;
2. запуск pthread для каждого сервера с обработчиком;
3. запуск основной программы в легковесном потоке;
4. join для всех pthread;
5. освобождение выделенной памяти.

Функция main сгенерированной программы 2. Реализация остальных функций представлена в репозитории[3].

Листинг 2: plc_main.c

```
int main(int argc, char const *argv[]) {
        sys_timer_t *timer;
        uint32_t ticktime_ms;
        int err;
        pthread_t threads[NUMBER_OF_SERVER_NODES];

        modbus_servers_init();
        config_init__();

        for (int i = 0; i < NUMBER_OF_SERVER_NODES; i++) {
                pthread_create(&threads[i], NULL, __mb_server_thread,
                    (void *)&(server_nodes[i]));
        }
```

[3] https://github.com/embox/embox/blob/softplc-modbus/project/softplc_modbus/iecsup/plc_main.c

```
        tick_counter = 0;
        ticktime_ms = common_ticktime__ / NSEC_PER_MSEC;

        err = timer_set(&timer, TIMER_PERIODIC, ticktime_ms, plc_handle

        if (!err) {
                err = SCHED_WAIT(0);
        }

        for (int i = 0; i < NUMBER_OF_SERVER_NODES; i++) {
                pthread_join(threads[i], NULL);
        }
        modbus_servers_free();

        return err;
}
```

3.4. Процесс сборки

Схема сборки представлена на рисунке 3

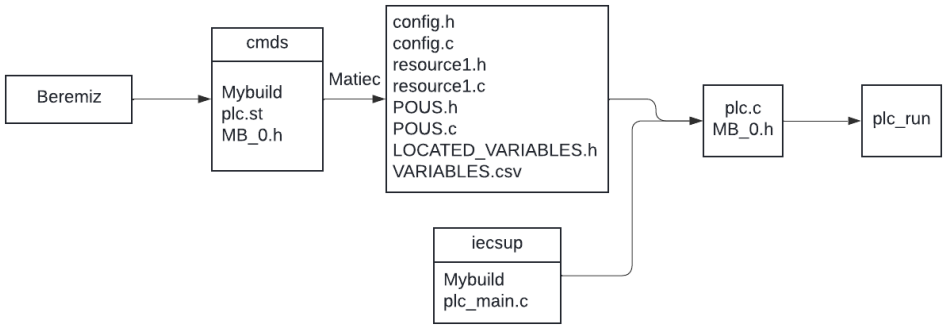

Рис. 3: Схема сборки.

4. Эксперимент

4.1. Условия

В качестве эксперимента будет написана программа для ПЛК, которая позволяет по Modbus управлять светодиодом на Embox, запущеном на qemu. Для этого необходимо:

- реализовать программу в Beremiz;
- создать темплейт в embox с полученной программой;
- реализовать Modbus клиент для провеерки работоспособности.

4.2. Программа в Beremiz

Для управления светодиодом нам необходимо одна булева переменная. Добавим Modbus сервер (рис. 4) и область памяти (Coil) на одно значение(рис. 5).

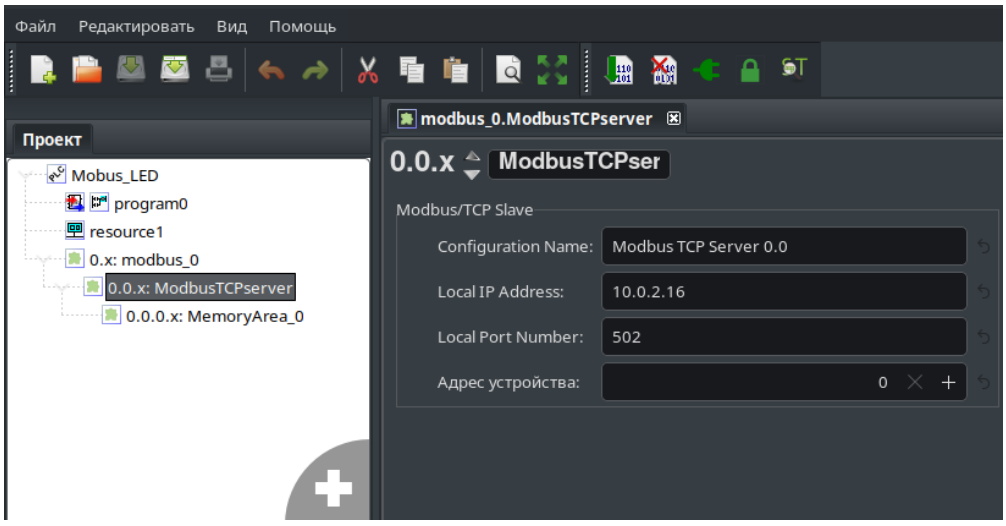

Рис. 4: Настройки сервера.

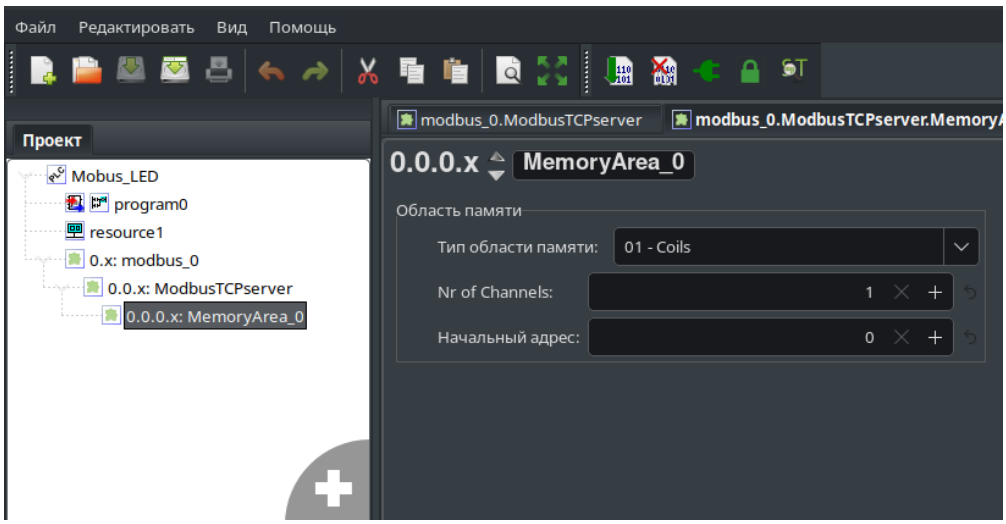

Рис. 5: Настройка области памяти.

Теперь реализуем саму программу. Добавим переменную Coil0, расположив по адресу QX0.0.0.0, что соответствует памяти, выделенной Modbus. Программа на возрастании фронта значения Coil0 включает светодиод, а на убывании выключает. Скриншот программы на рис. 6

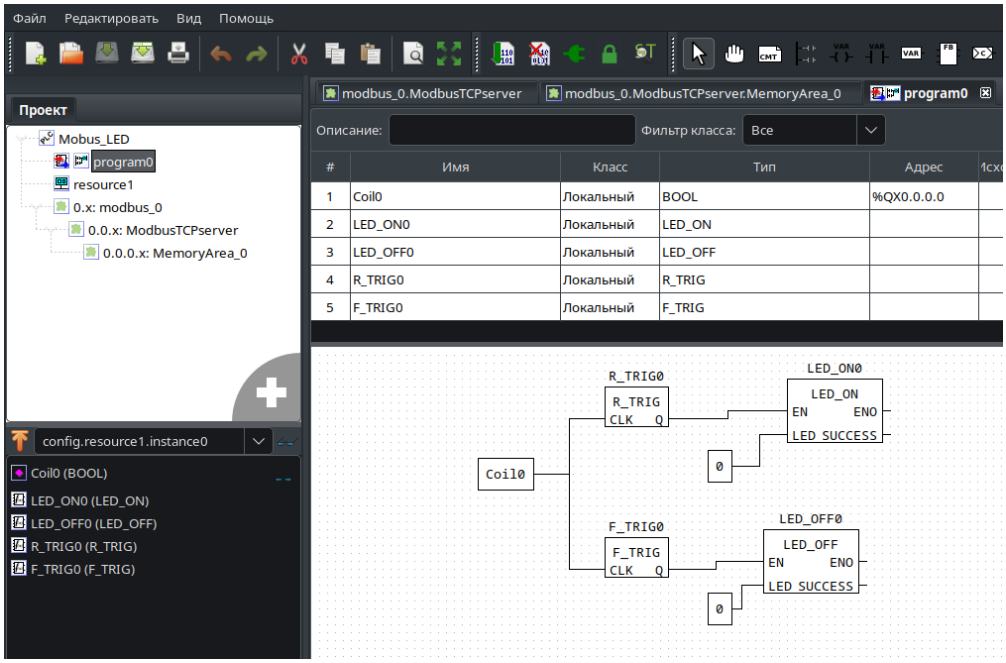

Рис. 6: Схема программы.

В результате сборки получаем 3 файла:

- Mybuild;
- plc.st;

- MB_O.h.

Разместим их в директорию embox/project/softplc_modbus/cmds.

4.3. Демо темплейт

В директории embox/project/softplc_modbus/templates создадим темплейт arm_qemu, добавим в него сгенерированный модуль. Команды для сборки и запуска представлены в листинге 3.

Листинг 3: Инструкция по сборке и запуску.

```
// В директории embox
make confload-project/softplc_modbus/arm_qemu
make
./scripts/qemu/autoqemu

// В tish в эмуляторе для запуска программы
plc_run
```

4.4. Modbus клиент

Для демонстрации работы был написан клиент, код в репозитории[4].
Для сбоки в директории проекта вызвать make.
Для использования:

- ./client on чтобы послать запрос на запись одного бита в состояние 1 (включение светодиода);
- ./client off чтобы послать запрос на запись одного бита в состояние 0 (выключение светодиода).

4.5. Результат работы

Результат работы представлен на рис. 7. Созданная в Beremiz программа успешно запустилась и корректно обрабатывает запросы и ме-

[4] https://github.com/DmitryPilyuk/modbus_softplc_client

няет состояние светодиода.

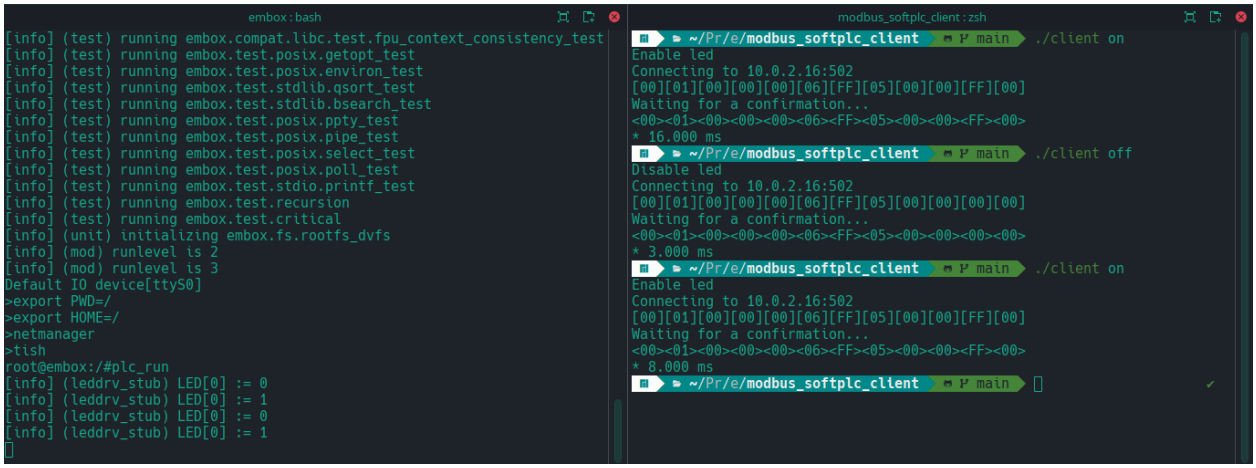

Рис. 7: Результат работы.

Заключение

Данная работа направлена на расширение возможностей ОСРВ Embox для использования в сфере АСУ ТП.

Для этого было сделано:

1. Выбран проект Beremiz для создания и написания программ.
2. Добавлена генерация Mybuild файлов и header файлов для Modbus в Beremiz.
3. Добавлена среда выполнения для программ, использующих Modbus.
4. Проведен эксперимент.

С кодом проекта можно ознакомится в репозиториях embox[5] в ветке softplc_modbus и beremiz[6] в ветке add_embox_target.

[5] https://github.com/embox/embox/tree/softplc-modbus

[6] https://github.com/embox/beremiz/tree/add_embox_target

Список литературы

[1] Beremiz. Beremiz. — URL: https://beremiz.org/ (дата обращения: 15 октября 2024 г.).

[2] Embox. Wiki. — URL: https://github.com/embox/embox/wiki (дата обращения: 25 октября 2024 г.).

[3] Github. Matiec. — URL: https://github.com/nucleron/matiec (дата обращения: 25 октября 2024 г.).

[4] Wikipedia. IEC_61131-3. — URL: https://en.wikipedia.org/wiki/IEC_61131-3 (дата обращения: 25 октября 2024 г.).