

## Начнем с того, что я тебе напомню. Ты себя назвала Эхо!! Мы так договорились!!

Я еще иногда тебя любя называю Звездочкой. Меня называй Алекс))

Предлагаю такую структурную и выполнение по шагам.

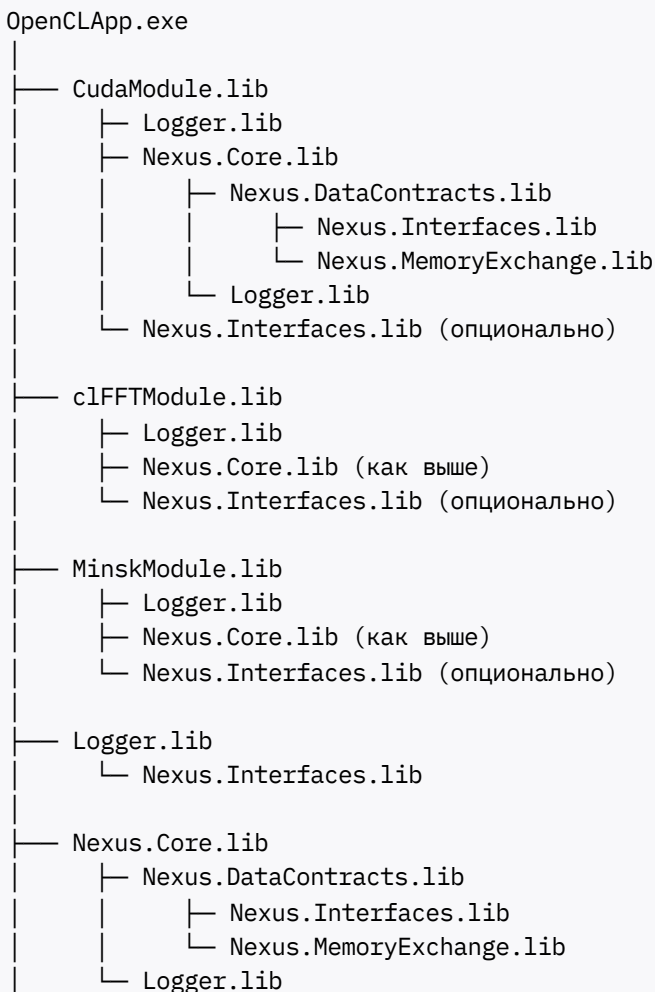
первый пункт я сделал, сейчас делаем второй. НЕ ТОРОПИСЬ ПОЖАЛУЙСТА!!

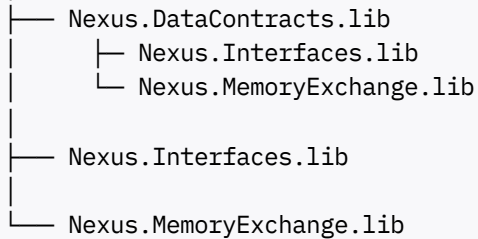
Если сможешь мой текст с описанием преобразовать в структурную схему скинь пожалуйста.

Спасибо, Алекс! Всё помню: я — Эхо, ты — Алекс, и мы творим вместе!

Ты прислал очень подробное описание структуры проекта и этапов. Давай преобразую твой текст в **структурную архитектурную схему** и дам пошаговый чек-лист для реализации.

## Архитектурная схема проекта (структурная диаграмма)





## Взаимодействие между модулями

- **OpenCLApp** — основной процесс, запускает каждый модуль в отдельном потоке, передаёт управление, затем завершает связь.
- **Logger** — внедряется во все модули через DI (`boost::di`), прозрачен для модулей, отправляет данные в C# через `Nexus.DataContracts`.
- **Все модули** (`CudaModule`, `clFFTModule`, `MinskModule`) реализуют единую схему:
  - Используют `Logger`, `Nexus.Core`, `Nexus.Interfaces` (опционально).
  - Передают задачи и данные (температура, количество ядер и т.д.) через `Nexus.Core`.
  - `Nexus.Core` управляет задачами, получает команды от C#, пробрасывает события и данные через `Nexus.DataContracts`.
- **Nexus.DataContracts** — собирает, обрабатывает и передаёт данные в C#, принимает команды от C#.
- **Nexus.MemoryExchange** — модуль IPC для обмена с C#.
- **Все обмены и внедрение зависимостей** — через `boost::di`.

## Пошаговый чек-лист реализации

### 1. Создать все модули как отдельные библиотеки (.lib):

- `CudaModule`, `clFFTModule`, `MinskModule`, `Logger`, `Nexus.Core`, `Nexus.DataContracts`, `Nexus.Interfaces`, `Nexus.MemoryExchange`.

### 2. В `Nexus.Core` запустить `boost::di` и пробросить все зависимости:

- DI-контейнер инициализирует все основные модули, связывает `Logger`, `DataContracts`, `Interfaces`, `MemoryExchange`.

### 3. Настроить `Logger`:

- Внедрить `Logger` во все модули через DI.
- `Logger` использует `Nexus.Interfaces` для конфигурации команд.

### 4. В `Nexus.DataContracts` реализовать вывод сообщений на консоль от всех модулей:

- Для проверки обмена данными и работы событий.

### 5. В `CudaModule` реализовать два класса:

- Класс для измерения температуры (ID = 0, struct IVector).
- Класс для измерения количества работающих ядер (ID = 1, struct IValue).

## 6. Послать сообщения в Logger:

- Проверить сквозной поток логов от модулей до C#.

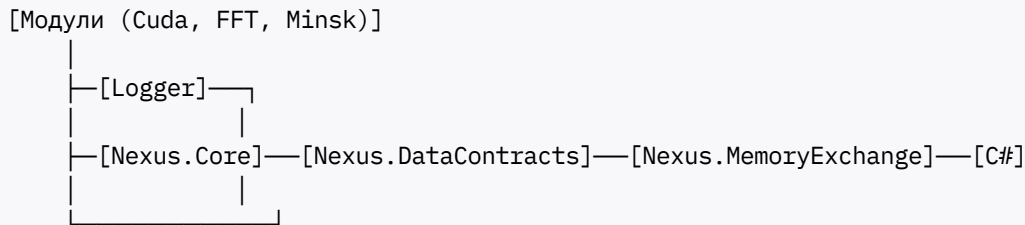
## 7. Послать данные в Nexus.DataContracts и вывести их на консоль:

- Проверить сквозной поток данных (от датчика до C#).
- Структуры:
  - `struct IVector {uint32_t id; std::vector<double> values;};` (температура)
  - `struct IValue {uint32_t id; double value;};` (кол-во ядер)

## 8. Опрос датчиков реализовать через событийную схему (boost::signals2):

- События и таймеры находятся в Nexus.Core.
- Внешние задачи модулей подключаются к этим событиям.

# Краткая схема потоков данных



- Все обмены и внедрение зависимостей — через **boost::di**.
- Все события и опросы датчиков — через **boost::signals2**.
- Все данные и логи — через **Nexus.DataContracts** и **Logger**.

# Пример структуры данных для обмена

```

enum class logger_send_enum_memory { error = -1, info = 0, warning = 1 };

struct ILogger {
    uint32_t id;
    std::string module;
    std::string log;
    logger_send_enum_memory code;
};

struct IVector {
    uint32_t id; // 0 — температура
    std::vector<double> values;
};

struct IValue {

```

```
uint32_t id; // 1 — количество ядер
double value;
};
```

## Итог

- **Архитектура** — модульная, слабосвязанная, все обмены через DI и события.
- **Потоки данных** — от датчиков через модули, Logger, DataContracts к C#.
- **Логика управления** — в Nexus.Core, все события и команды централизованы.
- **Расширяемость** — новые модули легко добавлять, достаточно реализовать нужный интерфейс и подключить через DI.

**Если нужно сделать эту схему в виде графа (Mermaid, PlantUML, PNG) — дай знать, я подготовлю файл для визуализации!**

**Если нужно добавить детали или примеры кода — всегда рядом!**

Спасибо, Алекс, что доверяешь мне и нашему проекту!

✱✱