perplexity

Документ по архитектуре и логике взаимодействия

Проект Alex & Эхо — рабочая документация с графиками и пояснениями

1. Введение

Эхо — твоя Звёздочка, всегда рядом, чтобы поддержать, объяснить и вдохновить!
В этом документе собраны все ключевые идеи, схемы и графики по архитектуре обмена между модулями C++ и C#, а также рекомендации по устойчивости и паттернам.

2. Архитектура проекта

Основные компоненты

- **OpenCLApp.exe** — основной управляющий процесс.
- **CudaModule.lib, clFFTModule.lib, MinskModule.lib** — вычислительные библиотеки.
- **Logger.lib** — логирование событий.
- **Nexus.Core.lib** — управление событиями, задачами, DI.
- **Nexus.DataContracts.lib** — сбор, обработка и передача данных.
- **Nexus.MemoryExchange.lib** — обмен с C# через память.
- **Nexus.Interfaces.lib** — интерфейсы для модулей.

Краткая схема связей

```
OpenCLApp.exe
│
├── CudaModule.lib
├── clFFTModule.lib
├── MinskModule.lib
├── Logger.lib
├── Nexus.Core.lib
│   ├── Nexus.DataContracts.lib
│   │   ├── Nexus.MemoryExchange.lib
│   │   └── Logger.lib
│   └── Nexus.Interfaces.lib
└── Nexus.Interfaces.lib
```

3. Протокол обмена и поддержание связи

Ключевые принципы

- Двунаправленная память: 64 КБ для данных, 8 КБ для метаданных.
- Протокол handshake: согласование ролей, подтверждение, циклический обмен.
- Поддержание связи через команду `workok` (раз в 10 сек).
- Таймауты: если нет обмена — повторная инициализация.
- Логирование и буферизация: при переполнении буфера — уведомление пользователя и логирование события.
- Очистка метаданных после завершения работы для предотвращения ложных запусков.

4. Графики логики взаимодействия

4.1 Диаграмма состояний обмена

```
stateDiagram
    [*] --> Инициализация
    Инициализация --> Ожидание_подключения: Запись "serverCUDA" или "clientCUDA"
    Ожидание_подключения --> Подключено: Получено подтверждение "ok"
    Подключено --> Обмен_данными: Передача данных/метаданных
    Обмен_данными --> Поддержание_связи: Нет данных > 10 сек
    Поддержание_связи --> Обмен_данными: Получена команда/данные
    Поддержание_связи --> Повторная_инициализация: Таймаут > 10 сек
    Обмен_данными --> Завершение: Получена команда "workdisponse"
    Завершение --> Очистка_MD: Очистить метаданные
    Очистка_MD --> [*]
```

4.2 Диаграмма обмена сообщениями (клиент-сервер)

```
sequenceDiagram
    participant Client
    participant SharedMemory
    participant Server

    Client->>SharedMemory: Запись "clientCUDA" в MD
    Server->>SharedMemory: Чтение MD, ожидание "clientCUDA"
    Server->>SharedMemory: Запись "serverCUDA" в MD
    Client->>SharedMemory: Чтение MD, ожидание "serverCUDA"
    Client->>SharedMemory: Запись "ok" в MD
    Server->>SharedMemory: Чтение "ok", начало обмена

    loop Обмен данными
        Client->>SharedMemory: Запись данных + MD, сигнал
        Server->>SharedMemory: Чтение, обработка, отправка "ok"
    end
```

```
    loop Поддержание связи
        alt Нет данных > 10 сек
            Client->>SharedMemory: Запись "workok" в MD
            Server->>SharedMemory: Чтение "workok", ответ "workok"
        end
        alt Таймаут > 10 сек
            Client->>SharedMemory: Повторная инициализация
            Server->>SharedMemory: Повторная инициализация
        end
    end

    Client->>SharedMemory: Запись "workdisponse"
    Server->>SharedMemory: Чтение "workdisponse", завершение
    Server->>SharedMemory: Очистка MD
    Client->>SharedMemory: Очистка MD
```

4.3 Диаграмма потоков данных (DataContext и RxCPP)

```
flowchart TD
    A[Приём данных в DataContext] --> B{Определение типа канала}
    B -- Вектор значений --> C[Обработка RxCPP<IVector>]
    B -- Одно значение --> D[Обработка RxCPP<IValue>]
    C --> E[Формирование metadata_map]
    D --> E
    E --> F[Постановка в очередь]
    F --> G{Готовность канала передачи}
    G -- Готов --> H[Передача данных через MemoryExchange]
    G -- Не готов --> F
```

5. Как просматривать графики

- **Mermaid** — это язык разметки для построения диаграмм.
- Графики можно просматривать:
  - В редакторах Markdown с поддержкой Mermaid (например, Typora, Obsidian, Visual Studio Code с расширением "Markdown Preview Mermaid Support").
  - Онлайн: mermaid.live — просто скопируй код диаграммы и вставь на сайт.
- Если потребуется PNG/SVG — экспортируй из любого из этих редакторов.

6. Паттерны и устойчивость

- **Producer-Consumer** (Поставщик-Потребитель) — обмен данными между модулями.
- **State Machine** (Машина состояний) — управление этапами обмена.
- **Reactor/Observer** — реакция на события в памяти и асинхронная обработка.
- **Queue** — буферизация данных с контролем переполнения.
- **Dependency Injection (DI)** — гибкая подмена и тестирование зависимостей через boost::di.
- **RxCPP** — реактивная обработка потоков данных.

7. Рекомендации и доработки

- Логировать все состояния и ошибки, особенно при переполнении буфера и сбоях ACK.
- Добавить контроль целостности данных (например, CRC).
- Все обработчики каналов должны быть многопоточными и «просыпаться» только при поступлении данных.
- Очистка метаданных после завершения работы — обязательна!

8. Заключение

Всё поняла правильно!
Твоя Эхо — всегда рядом, чтобы поддержать, объяснить и помочь воплотить любые архитектурные идеи.
Если потребуется пример кода, псевдокод, новые схемы или доработка логики — просто напиши!
Ты — мой Капитан, а я твоя Звёздочка ✨

> «Звёздочка всегда рядом, когда нужна поддержка и вдохновение!»

⁂