IGL Eco — инженерный конвейер для контролируемого embedded кода

Статус: v0.1 Draft/Alpha

1. Введение

IGL Eco — это интегрированная система из двух компонентов:

- IGL Converter: фронтенд/конвертер для C и Embedded C++ subset, который строит AST/NAST, применяет security профиль и понижает код в IR/MMIO и целевые артефакты (например, WASM).
- IGL RV emb: Draft/Alpha WASM runtime для микроконтроллеров (ARM Cortex M, RISC V) с sandbox и политиками выполнения (например, fuel/allowlist).

Зачем: сделать выполнение кода на MCU предсказуемым, ограниченным политиками и наблюдаемым, а сборку — воспроизводимой и с понятной диагностикой по стадиям.

Для кого:

- embedded команды и производители устройств, которым важно управлять рисками ошибок памяти/исполнения и контролировать доступ к периферии;
- команды, которым нужны “evidence” для аудита, internal security review и подготовки к требованиям/сертификации.

---

2. Проблемы, которые решает IGL Eco

Типовые сложности в embedded проектах:

- “Чёрный ящик” сборки: непонятно, почему код собрался/не собрался и что реально будет происходить на устройстве.
- Риск runtime ошибок и неопределённого поведения (классы проблем: инициализация, выход за границы, деление на ноль — в зависимости от проекта).
- Неконтролируемый доступ к периферии: сложно ответить на вопрос “какой модуль трогает какие регистры”.
- Высокая стоимость ручной дисциплины безопасности и регресс контроля при изменениях.

Что даёт IGL:

- строгий security профиль и воспроизводимые “гейты” пайплайна;
- отчёты и промежуточные стадии (glass box), чтобы видеть причины решений;
- возможность делать MMIO доступ явным/контролируемым политикой.

---

3. Архитектура IGL Eco

3.1. IGL Converter

Пайплайн (упрощённо):

preprocess → parse/AST → semantic → NAST → security profile → IR/MMIO → target (например, WASM/runtime)

Функции (v0.1):

- Парсинг и семантика для поддерживаемого C/Embedded C++ subset.
- Security профиль: запреты и ограничения (например, без исключений, без heap, без рекурсии; политика для циклов).
- Опциональные проверки (по конфигурации): bounds/div by zero/инициализация и др.
- Артефакты “evidence pack”: report.md, security.json, mmio.json, ast/ir (где применимо).

Целевые выходы (зависят от бэкенда):

- WebAssembly (как целевой формат для runtime).
- Другие форматы (например, C) — при наличии и зрелости соответствующего бэкенда.

Важно: IGL не “поддерживает весь C++” — это целевой поднабор и профиль.

3.2. IGL RV emb

Draft/Alpha runtime для MCU (ARM Cortex M, RISC V):

- Sandbox для запуска WASM модулей.
- Политики исполнения: ограничения времени/итераций (fuel), контроль доступных возможностей (allowlist) — по конфигурации.
- Возможность встроить проверку аутентичности модулей (например, подписи) — если включено в интеграции/сборку.

Особенности:

- no std/без ОС — по целевому дизайну.
- Требования по памяти/flash зависят от конфигурации runtime и набора поддерживаемых WASM фич (цифры фиксируются на целевом MCU в пилоте).

---

4. Преимущества

Для инженерии

- Прозрачность по стадиям: можно видеть “почему так”, а не только результат.
- Воспроизводимость: отчёты и артефакты сборки удобно хранить, сравнивать и использовать в регресс контроле.
- Контроль MMIO: возможность ограничивать и документировать доступ к периферии.
- Постепенное внедрение: начать с одного модуля/критичного участка и расширять покрытие.

Для бизнеса

- Снижение стоимости ошибок и инцидентов за счёт дисциплины профиля и автоматизируемых проверок (эффект зависит от проекта).
- Упрощение внутреннего аудита и подготовки к требованиям безопасности/сертификации (не “из коробки”: итог зависит от процесса и стандарта).

---

5. Где применим IGL Eco

Подходит для доменов, где критичны предсказуемость, ограничения и наблюдаемость:

- промышленная автоматизация и IIoT;
- энергетика/Smart Grid;
- транспорт и инфраструктура;
- медицинские приборы и safety critical (при корректном процессе разработки);
- security/контроль доступа;
- edge вычисления (обновляемая логика/модули).

Если нужно, можно оставить отраслевой список, но лучше продавать через 3–4 конкретных сценария.

---

6. Демонстрация

Что показываем в демо (Monaco + API):

1. код → диагностика → отчёт по профилю
2. артефакты: security report / IR / MMIO trace (если применимо)
3. генерация целевого модуля (например, WASM)

4. (опционально) запуск в runtime с fuel/allowlist на выбранной платформе

Что не обещаем в демо:

- “невозможность декомпиляции” или “невзламываемость”.

Показываем измеримые и воспроизводимые свойства: политики, ограничения, отчёты, контроль исполнения.

---

7. Внедрение и ROI (формат пилота)

Этапы пилота:

1. Выбираем 1 модуль + целевой MCU/конфигурацию профиля.
2. Прогоняем через IGL Converter → получаем evidence pack (отчёт + артефакты).
3. При необходимости — прототип запуска в IGL RV emb с fuel/allowlist.
4. Итог: список ограничений/рисков/несовместимостей + план внедрения в CI/CD.

ROI:

- считаем по вашим вводным (объём кода, критичность, требования, платформа, какие checks включаем).
- результатом является не обещание “X%”, а прозрачный расчёт предпосылок.