Актуальность

По мере продвижения разработки проекта становится понятно, что после завершения работы над платформой разработки (devkit) и созданием прототипа на её основе дальнейшей развитие должно идти в направлении создания полноценного проприетарного устройства, реализующего весь необходимый функционал, отвечающего всем требованиям к отказоустойчивости и ремонтопригодности.

Данный документ предназначен для оценки возможных путей и способов достижения данной цели.

Цели

Основной конечной целью всей происходящей и будущей деятельности является создание проприетарной аппаратной и программной платформы, предназначенной для интеграции в автомобили с целью сбора, анализа и передачи различной информации, а также использования этой информации для реагирования на различные внештатные ситуации в реальном времени.

Данная цель достижима путём выполнения нескольких “подпроектов”, каждый из которых покрывает определённую часть работ

1. **R&D (Исследование и разработка)**
   Текущий этап. Подразумевает создание и тестирования грубой версии итогового устройства с целью валидации общей концепции и инженерных решений, и частичной разработки АО и ПО, с целью использования всех полученных наработок на следующих этапах.
   Результатом данного этапа является создание полноценной платформы для разработки (devkit), набор опыта по работе с платформой и выявление её особенностей, а также разработка сопровождающих материалов для дальнейшей работы. На данном этапе девкит уже способен частично или полностью выполнять весь функционал итогового устройства, но не готов к массовому производству и эксплуатации в реальных условиях ввиду несоответствия всем требованиям отказоустойчивости АО и ПО.
2. **Development (Разработка)**
   Следующий этап работы, на котором и будет сфокусирован дальнейший материал. Подразумевает использование наработок, полученных во время R&D, с целью создания полноценной версии устройства, отвечающей всем требованиям и стандартам, и готовой к запуску в производство и использование в реальных условиях.
   Результатом данного этапа будет выход аппаратной и программной частей на версию 1.0.0 согласно семантическому версионированию, а также подготовка полного пакета документации, необходимого для изготовления, проверки качества, установки и обслуживания конечного устройства. Если требуется, то в

данный этап также могут быть включены прохождения тестов и получение сертификаций о соответствии устройства государственным и общепринятым корпоративным стандартам.
По окончанию этого этапа разработка устройства может считаться полностью завершенной.

3. **Поддержка и обслуживание**
   Данный этап начинается после запуска производства и введения устройств в эксплуатацию, в ходе которой неизбежно будут выявлены ошибки в ПО, а также будут выделены возможности для улучшения и доработки. Помимо этого возможно также возникновение внештатных ситуаций во время изготовления или установки устройства (например, вывод из производства того или иного компонента).
   Решение данных сложностей заключается в наличии постоянной деятельности по поддержки АО и ПО в актуальном и работоспособном состоянии, что в свою очередь будет заключаться в формировании некой постоянной команды поддержки и обслуживания устройства.
   Данный этап подразумевает постоянную реактивную постановку и выполнение задач, и завершается полностью лишь после вывода устройства из эксплуатации.

Задачи

R&D

Задачи данного этапа уже сформулированы и описаны в имеющейся документации, но свести всю работу можно к нескольким ключевым этапам

1. Анализ требований и имеющихся решений
2. Анализ возможных способов и инструментов реализации нового решения
3. Создание прототипа решения
4. Тестирование в реальных условиях
5. Документирование и анализ результатов тестов
6. Составление плана дальнейшей работы

Разработка

Данный этап будет более продолжительным и на данный момент времени невозможно точно предсказать, какие точные этапы потребуются для выполнения поставленных целей, но примерный состав работ будет выглядеть следующим образом.

Каждый из этапов разработки также можно рассматривать как подпроект

Pre-production (подготовка)

На данном этапе будет проведён анализ всех предыдущих работ, выделены и зафиксированы выявленные недочёты, а так же обозначены работы, необходимые для выхода на версию 1.0.0 (далее - V1).

Результатом этапа будет пакет документации, описывающий весь спектр запланированных работ, необходимых закупок и сроков реализации V1.

Расширенной целью данного этапа является частичное/полное выполнение “списка покупок”

- Определение спецификаций конечного устройства с заказчиком
    - Запрос требований заказчика | 5-10 часов
    - Кросс-командный анализ требований, поверхностный анализ зон ответственности и способов реализации | 10-20 часов
    - Анализ требований внутри команд согласно зонам ответственности, подготовка документации по способам реализации | 30-40 часов
    - Кросс-командный анализ предложенных решений, предварительный отбор итогового способа реализации по каждому пункту | 10-20 часов
    - Презентация заказчику возможных способов реализации, получение ОС | 5-10 часов
    - Формирование ТЗ с итоговым функционалом и способами реализации | 20-30 часов
    - Подтверждения ТЗ с заказчиком | 10-20 часов
- *Опционально:* Подготовка инфраструктуры
    - Анализ требований к инфраструктуре проекта | 10-20 часов
    - Подготовка АО инфраструктуры | 30-40 часов
    - Развёртывание системы базы знаний для документации и менеджмента | 10-20 часов
    - Развёртывание системы контроля версий | 10-20 часов
    - Развёртывание системы постоянной сборки и интеграции (CI/CD) | 20-30 часов
    - Развёртывание системы мониторинга инфраструктуры | 20-30 часов
    - Развёртывание системы резервного копирования | 20-30 часов
    - Документирование проделанной работы и использованных решений | 5-10 часов
- Сбор и анализ результатов предыдущих работ внутри команд
    - Поиск и фиксация недочётов и недоработок, которые необходимо исправить | 10-20 часов
    - Проверка на наличие случаев “не баг, а фича” с заказчиком | 10-20 часов
      Т.е. проверка на использование ошибок/непредусмотренных функций в эксплуатации устройства во избежание их удаления/изменения.
    - Анализ и документирование способов решения проблем с учётом ТЗ | 20-30 часов
      По принципу оставить-рефакторить-переделать
- Планирование следующего этапа (production)
    - Планирование с заказчиком периодичности и формата отчётности | 5-10 часов

    - Составление мастер-листа задач в виде канбан-доски, расставление приоритетов, зависимостей и ответственности | 20-30 часов
    - Оценка задач в человеко-часах внутри команд | 10-20 часов
    - Определение необходимого АО и ПО, которое необходимо закупить | 10-20 часов
    - *Опционально:* Подготовка “промо-материалов” для аргументирования закупок | 20-30 часов
    - Презентация запроса заказчику, определение “списка покупок” | 5-10 часов
    - Дополнение ТЗ “списком покупок” | 5-10 часов
    - Разбиение списка задач на спринты с учётом сроков получения АО и ПО | 10-20 часов
    - Утверждение плана работ с заказчиком | 10-15 часов

Production (разработка)

На данном этапе будут непосредственно осуществляться задачи по реализации, определённые во время pre-prodction.

Важное замечание: Ввиду невозможности предсказать результаты всех этапов между текущим моментом и началом разработки V1 соответствие представленного ниже описания тому, что будет сформировано в итоге, не гарантируется, и, вероятно, будет изменено.

- Кросс-командная разработка стандартов и интерфейсов
    - Определение требований зон ответственности внутри команд | 10-20 часов
    - Обмен взаимными требованиями и ограничениями | 10-20 часов
    - Документирование интерфейсов | 20-30 часов
        - Конфигурация CI/CD для автоматической сборки и тестирования | 30-40 часов
    - Интеграционное тестирование заглушек интерфейсов | 20-30 часов
    - Сверка сходства целевого функционала и реализации | 5-10 часов
- Внутрикомандная работа над зонами ответственности
    - **Команда АО**
      Команда, работа которой сфокусирована на разработке аппаратного обеспечения устройства: схемотехники модулей и соединений, охлаждения, корпусов и прочих аспектов, непосредственно связанных с физическими параметрами устройства. Тесно работает/пересекается с командой системного ПО.
        - Детальная проработка модулей, выделенных во время pre-production | 80-120 часов
        - Поиск исполнителя для разводки печатных плат модулей (при условии заказа разводки) | 20-30 часов
        - Для каждого модуля (140 - 230 часов)
            - Подготовка и оформление требований к печатной плате модуля | 40-60 часов

BOM, допуски, размеры, расположение крепёжных отверстий и т.д.
        - Подготовка документации для заказа разводки плат | 20-30 часов
        - Работа совместно с исполнителем в процессе разработки | 10-20 часов
        - Валидация прототипов | 40-80 часов
          Индивидуальная валидация (тестирование самого модуля вне системы)
          Интеграционная валидация (тестирование в уже имеющемся прототипе системы)
        - Документирование прототипов | 30-40 часов
    - Интеграционное тестирование АО | 60-80 часов
      Конфликты радиосигналов, нестабильные соединения, устойчивость к воздействию и т.д.
    - Документирование АО | 40-60 часов
- **Команда системного ПО**
    - Разработка ПО мультиконтроллера платы-носителя ЦВМ | 80-120 часов
    - Разработка системы UOTA (Unattended Over-the-Air) обновлений | 80-100 часов
    - Разработка централизованной системы логирования | 40-60 часов
    - Разработка системы сбора и передачи телеметрии | 80-100 часов
    - Разработка инструментов для диагностики и отладки | 40-80 часов
    - Разработка системы самодиагностики/самовосстановления | 60-80 часов
    - Разработка системы горячей замены хранилища | 40-80 часов
    - Разработка системных пользовательских интерфейсов | 80-100 часов
    - Для каждого модуля АО (150 - 310 часов)
        - Разработка эмулятора модуля согласно разработанной ранее документации | 30-60 часов
          Необходимо для ускорения работы во время ожидания прототипов модулей и разработки тестов
        - Разработка автоматизированных тестов | 20-30 часов
        - Анализ совместимости актуальной реализации модуля с уже имеющимся ПО | 10-20 часов
        - Разработка/актуализация ПО модуля с учётом изменений в АО | 10-80 часов
        - Разработка/актуализация документации ПО модуля | 10-20 часов
        - Актуализация эмулятора | 10-20 часов
    - Анализ информационной безопасности | 10-20 часов
    - Корректировка уязвимостей | 0-30 часов
    - Интеграционное тестирование ПО | 40-60 часов
    - Нагрузочное тестирование ПО | 20-40 часов
- **Команда(ы) пользовательского ПО**
  Для каждой команды точные шаги разработки самого приложения

будут отличаться, поэтому тут описаны и оценены этапы, которые будут присутствовать в разработке любого приложения под платформу.
    - Ознакомление с документацией на АО/системное ПО, консультации с соотв. командами | 5-10 часов
    - Разработка ПО | ??? часов (см. описание выше)
    - Интеграционное тестирование | 5-10 часов/спринт
    - Анализ информационной безопасности | 10-20 часов
    - Корректировка уязвимостей | 0-30 часов
- Промежуточная интеграция
  Данная группа работ вынесена в отдельный пункт, но фактически будет происходить на регулярной основе, чтобы гарантировать совместимость и реализацию функционала по завершению работ.
    - Внутрикомандная сверка реализации с документацией интерфейсов | 5-10 часов
    - Внесение необходимых изменений в реализацию или документацию | 0-10 часов
    - Кросс-командная интеграция и валидация решений | 10-20 часов
    - Исправление обнаруженных ошибок и несостыковок | 10-20 часов
- Финальная интеграция
    - Кросс-командная проверка совместимости | 5-10 часов
    - Поэтапная интеграция всех систем | 20-30 часов
    - Интеграционное функциональное тестирование ПО | 30-40 часов
    - Интеграционное нагрузочное тестирование ПО | 30-40 часов
    - Анализ информационной безопасности | 20-40 часов
    - Корректировка уязвимостей | 0-40 часов
    - Предварительная документация errata | 10-20 часов
- Внутренняя валидация итогового решения
    - Создание тестового стенда | 10-20 часов
      Статического, или динамического на базе авто
    - Тестирование основных сценариев использования | 30-40 часов
    - Тестирование работы в условиях внешнего воздействия | 30-40 часов
    - Тестирование с участием внешних тестировщиков | 20-30 часов
    - Корректировка выявленных недочётов | 20-40 часов
- Получение сертификаций
    - Определение потребности в получении сертификаций, составление списка необходимых сертификаций | 10-20 часов
    - Подготовка документации для сертификаций | 5-10 часов/серт.
    - Прохождение сертификаций | ??? часов
- Подготовка пакета документации, необходимой для эксплуатации устройства
    - Актуализация errata | 10-20 часов
    - Перепроверка и актуализация документации для разработчиков | 30-40 часов
    - Разработка инструкций для установщиков | 40-80 часов
    - Разработка пользовательских инструкций | 80-100 часов
- Приёмо-сдаточные работы
    - Подготовка презентационных материалов | 30-40 часов
    - Презентация результатов | 5-10 часов

    - Передача результатов | 10-20 часов

Поддержка и обслуживание

Этап начинается после окончания разработки и продолжается до вывода системы из эксплуатации (End of Life/EOL)

- Подготовка команды поддержки и обслуживания
    - Формирование команды
    - Обучение команды
- Мониторинг системы
    - Отслеживание ошибок
    - Ручное тестирование
- Актуализация и обслуживание АО
    - Актуализация BOM АО с учётом изменения доступных компонентов
    - Актуализация ПО в соответствии с изменениями в АО
    - Тестирование новых партий/итераций компонентов АО
    - Ремонт повреждённых устройств
- Актуализация и обслуживание ПО
    - Исправление выявляемых в процессе эксплуатации ошибок
    - Отслеживание/актуализация информационной безопасности системы
    - Добавление новых функций
- Актуализация документации

Инженерные решения

Общее описания

Специфика разрабатываемого устройства подразумевает постоянную продолжительную эксплуатацию в агрессивных условиях с большими промежутками между ТО, если таковое вообще возможно. Это означает, что итоговое устройство должно не только реализовывать весь желаемый функционал, но и делать это с высоким уровнем надёжности и устойчивости к различным видам внештатных ситуаций. Представленные далее идеи призваны сделать аппаратное и программное обеспечение устройства более устойчивыми к различным неполадкам, и снизить время и затраты на обслуживание.

Программное обеспечение

Централизованное логирование и сбор телеметрии

Для отслеживания состояния системы, сборки аналитических и отладочных данных применяются журналы событий, или же логи (от анг. log - журнал). В используемой на платформе ОС Linux обычно используются несколько независимых систем

логирования событий самой ОС, а большая часть пользовательского ПО реализует свою собственную систему журналов, как правило на основе записи текстовых сообщений в файлы.

Для целей нашей платформы наилучшим решением будет создание единой централизованной системы журналов событий, которая будет собирать системные журналы и реализовывать функционал журналов для разрабатываемого нами ПО.

Преимущества

- Избавляет от необходимости реализовывать отдельный экспортёр логов на единый ЦОД системы под каждый журнал
- Упрощает и стандартизирует разработку пользовательского ПО; избавляет от необходимости реализовывать систему логирования для каждого приложения с нуля, и затем интегрировать её во все остальные системы
- Упрощает архитектуру
- Увеличивает видимость аномальных событий
- Увеличивает надежность, позволяя быстрее и точнее выявлять аномалии

Недостатки

- Необходимость поддержки ещё одного элемента ПО
- Наличие единой точки отказа системы журналирования на устройстве

Каскадные автономные обновления

Любое ПО требует периодического обслуживания: исправление неизбежно возникающих ошибок, закрытие уязвимостей (особенно в системах, имеющих подключение к Интернету), расширение функционала и т.д. Все операции подобного типа совершаются с помощью создания и установки обновлений. Как правило, данные обновления создаются разработчиками и устанавливаются вручную на целевые устройства, путём физического или удалённого подключения. Данный способ является самым распространённым и активно используемым в сфере embedded-разработки, но имеет ряд недостатков:

- Необходимость ручной сборки и тестирования любых обновлений, будь то небольшой патч безопасности или же глобальное обновление системы
- Высокая задержка между выходом новой версии ПО и её установкой в рамках обновления
- Высокая сложность оперативного обновления в случае выявления критической ошибки/уязвимости

Суммируя, такой подход является **сложным, времязатратным, медленным и рискованным**

Для решения этих проблем предлагается реализовать **автоматизированную систему постоянной сборки и тестирования обновлений с каскадным распространением**.

Данное определение подразумевает под собой систему, которая в автономном режиме проверяет наличие доступных обновлений ПО, собирает на их основе пакет обновления системы, тестирует его, и затем постепенно распространяет на небольшие группы устройств, чтобы отследить их поведение после установки обновления и предотвратить глобальный коллапс системы в случае возникновения ошибок, не выявленных на этапе автоматизированного тестирования. И в случае обнаружения таковых, уже обновлённые устройства могут быть автоматически возвращены на стабильную версию ПО за счёт **A/B структуры разделов**, а установка на ещё не обновлённые устройства будет отменена.

Каскадный подход также будет полезен для A/B тестирования (не связано со структурой разделов) различных функций на группах устройств, с целью проверки гипотез и эмпирического поиска оптимальных решений

<возможно тут добавить алгоритмы для различных ситуаций>

Единственным недостатком такой системы являются бóльшие инвестиции на этапе первоначальной разработки. Но в долгосрочной перспективе она позволит значительно сэкономить на обслуживании инфраструктуры за счёт автоматизации большинства регулярных задач по обновлению системы, а также значительного повышения отказоустойчивости и информационной безопасности за счёт поддержания в актуальном состоянии системы безопасности и снижения вероятности глобального отказа в случае неудачного обновления.

A/B структура разделов

Стандартная структура разделов подразумевает наличие одной копии каждого из разделов файловой системы на ПЗУ. Это решение является достаточным для большинства задач.

Но для наших задач более оптимальным решением будет так называемая “A/B” структура разделов. Она подразумевает дублирование нескольких/всех разделов ОС на ПЗУ. Данная структура имеет два основных преимущества: возможность обновления системы без перезагрузки и реализация резервной копии ПО.

Обновление системы без перезагрузки в нашем случае будет полезным, т.к. в большинстве ситуаций система будет активна только в промежутках времени, когда от неё требуется активная работа по назначению, что лишает нас возможности обновить ПО, процесс обновления которого требует перезагрузки. С A/B структурой мы имеем возможность произвести все необходимые операции по обновлению на группе разделов B во время работы из разделов группы A, после чего потребуется лишь изменить целевую группу разделов в загрузчике и быстро перезагрузиться в любое удобное время (например, перед переходом в спящий режим на время простоя авто)

Из A/B структуры разделов также вытекает постоянное наличие заведомо рабочей копии ПО на устройстве, что значительно повышает отказоустойчивость и снизит необходимость ручного обслуживания. Это достигается за счёт реализации системы обнаружения ошибок, которая будет определять различные внештатные ситуации, и в

случае выявления полностью откатывать всю систему к предыдущей стабильной версии.

Примером такой ситуации может стать неудачное обновление, которое в результате нарушения техпроцесса обновления или ошибки в самом ПО привело к невозможности запуска системы. При стандартной структуре разделов единственным выходом из такой ситуации будет ручная работа с устройством, вероятнее всего в виде перепрошивки, что вызовет простой системы и потребует трудозатраты со стороны специалиста по обслуживанию системы.
Но за счёт A/B структуры и системы самовосстановления в подобных ситуациях устройство сможет распознать внештатную ситуацию и восстановить нормальную работу полностью автономно.

Основными недостатками данной системы можно считать увеличенные требования к хранилищу и необходимость времени разработки на её реализацию. Стоит учесть, что первый пункт не является значительной проблемой, т.к. ПО устройства не займёт значительное пространство в ПЗУ и вряд ли потребует увеличения объёма последнего даже при полном копировании разделов.

Самодиагностика и самовосстановление

Невозможно спорить с тем, что вне зависимости от качества реализации АО и ПО внештатные ситуации в процессе эксплуатации являются неизбежными. Но приостановка использования и выделение времени специалиста на диагностику и обслуживание является крайне нежелательным способом решения возникающих неполадок ввиду высокой стоимости простоя и времязатрат специалиста. Поэтому рациональной будет реализация системы, которая позволит устройству самостоятельно оценивать своё состояние, выявлять неполадки, уведомлять о них и принимать меры по их автономному исправлению.

Данная система уже упоминалась в предыдущих пунктах, что обусловлено её глубокой интеграцией в ПО системы с целью отслеживания происходящих событий и выявления аномалий, на основании данных о которых будет приниматься решение о внештатном состоянии системы и возможных шагах для его разрешения.

Примером такой ситуации может стать обнаружение нарушение работы пользовательского ПО. Данное ПО планируется запускать в Docker, который уже имеет инструменты для обнаружения и разрешения внештатных ситуаций, но его возможности в данном направлении ограничены. Например, если в одном из контейнеров возникнет критическая ошибка, требующая переустановки контейнера из образа, то стандартных инструментов Docker не хватит, и тут в дело вступит система диагностики, которая распознает внештатное состояние (например, циклический перезапуск), и после стандартных попыток ручного перезапуска сможет полностью переустановить контейнер, восстановив нормальную работу до выхода обновления, решающего проблему или полноценного ТО системы.

API для доступа и управления ресурсами АО

Специфика разрабатываемого устройства подразумевает, что пользовательское ПО может, и, вероятно, потребует доступ к данным, предоставляемым АО, например, данные о местоположении, ускорении, времени по GPS, отправке SMS и т.д. Для реализации этого доступа можно использовать один из двух подходов: на стороне пользовательского ПО или на стороне системного ПО.

Первый способ

Первый способ подразумевает, что со стороны системы мы предоставляем полный и неконтролируемый доступ к АО, и все обязанности по работе с ним полностью ложатся на пользовательское ПО.

Преимущества

- Уменьшение задержки обновления данных
- Снижение нагрузки из-за отсутствия промежуточного слоя
- Возможность реализации интересных решений, требующих прямого доступа к АО

Недостатки

- **Крайне высокий** риск возникновения критической внештатной ситуации, нарушающей работу системы
- **Крайне высокий** риск нарушения информационной безопасности
- Необходимость повторной реализации работы с АО в каждом элементе пользовательского ПО

Второй способ

Второй способ подразумевает наличие промежуточного интерфейса, который предоставляет контролируемый доступ и готовые решения для работы с АО для пользовательского ПО.

Преимущества

- Наличие контроля над совершаемыми операциями; предотвращение опасных/несанкционированных действий
- Упрощение разработки пользовательского ПО; наличие готовых инструментов

Недостатки

- Увеличенная задержка обновления данных
- Дополнительная нагрузка на АО

Анализ обоих методов показывает, что первый способ является более быстрым, но **значительно** более рискованным. Учитывая, что в данной ситуации большинство приложений не потребуют высокой скорости обновления, а нагрузка и задержка от промежуточного слоя будут незначительными, первый способ с прямым доступом **крайне не рекомендуется.**

Второй способ, в свою очередь, потребует разработки программного интерфейса (API) для реализации доступа пользовательского ПО к АО. Данный интерфейс будет предоставлять доступ к данным, получаемым с АО, а так же доступ к различным методам управления АО. Например, активация линии тревожного сигнала, или возможность перезагрузки того или иного модуля в случае получения с него некорректных данных.

Данный API будет сопровождаться документацией и готовыми зависимостями для упрощения работы с ним в пользовательском ПО.

Удалённое управление (C2)

Даже при наличии всех предложенных здесь решений, призванных снизить необходимость участия людей в работе с системой, ситуации, когда прямое взаимодействие потребуется всё ещё не исключены. Реализовать прямой доступ можно несколькими способами, но основными являются прямой доступ к каждому устройству и управление через централизованный сервер.

Прямой доступ

Данный способ подразумевает наличие некого сервиса, предоставляющего доступ, запущенного на самом устройстве (сервере), к которому в последствии подключается устройство, используемое для обслуживания (клиент).

Для реализации доступа таким образом с точки зрения ПО достаточно будет использовать стандартные инструменты (ssh), но потребуются достаточно специфические инструменты с точки зрения сетей, т.к. каждое устройство в сети должно будет иметь свой уникальный IP адрес, что является редкостью в мобильных сетях, и обычно требует дополнительных постоянных расходов для поддержания.

Централизованный сервер

Способ подразумевает наличие некого централизованного сервера управления (Comand & Control/CnC/C2), к которому будут подключаться устройства для получения команд и передаче результатов их выполнения. Данный подход не требует особых условий с точки зрения сети, т.к. для него используется только исходящее подключение с устройства, но для корректной реализации потребуется проприетарное ПО для сервера и устройств, чтобы реализовать доступ безопасным образом, т.к. нарушение безопасности в данной ситуации может привести к значительным временным потерям и угрозе конфиденциальности.

Итоговое решение будет зависеть от возможностей, предоставляемых оператором мобильной связи. Если имеется возможность реализовать изолированную подсеть для этих устройств с возможностью ограниченного доступа извне, то оптимальным будет первый вариант.

Если же такой возможности нет или требуется высокая степень универсализации и отсутствие привязки к конкретной инфраструктуре оператора, то оптимальным будет второй вариант.

Аппаратное обеспечение

Захват и обработка видеопотока

Требования со стороны заказчика включают в себя необходимость поддержки возможности захвата различных аналоговых видеосигналов (AHD/TVI/CVI). Данная задача реализуема, но имеет ряд потенциальных сложностей, связанных с обработкой аналоговых сигналов.

Анализ ресурсов показал, что платформа RK3588 имеет встроенный ISP (Image Signal Processor) на 48 (24 * 2) мегапикселей, а также 2 * MIPI-CSIx4 (4 DPHY/3 CPHY) интерфейса для камер. Но для подключение аналоговой камеры к CSI интерфейсу требуется соответствующий чип-адаптер, и как показал первоначальный поиск, качественного решения для такой задачи нет. В теории, возможно разработать собственное решение на базе ПЛИС (FPGA), но оно будет значительно дороже и сложнее в долгосрочной поддержке, чем альтернативные варианты.

Поэтому оптимальным решением для задачи будет использование карты захвата аналогового видео, подключаемой к системе по USB/PCIe. Такие решения являются достаточно распространёнными, но не слишком хорошо задокументированными, поэтому на данный момент удалось обнаружить примеры таких карт захвата, но не их схемотехнику и BOM.pСудя по происходящему на изображении, данная плата содержит два чипа: один для обработки и первичного преобразования видеосигнала, и вторая, для его захвата, пост-обработки и передачи по USB. Для передачи данных такие карты используют стандартный USB 2.0/3.0, что значительно упрощает разработку модуля карты захвата для конечного устройства.

На данный момент неизвестно, какая микросхема используются в данных картах захвата, но очень вероятно, что именно она станет решением вопроса захвата видеопотока в нашем устройстве. Но пока что нет информации о точных аппаратных характеристиках, требованиях и реальной производительности, так что вполне вероятно, что придётся искать альтернативное решение.

Стоит также учесть, что для многих (всех найденных на данный момент) карт захвата производитель предоставляет возможность кастомизации платы и объёмных заказов, что также может стать допустимым способом реализации модуля карты захвата.

Существуют и более законченные решения вроде Advantech DVP-7036HE, которые остаётся только купить и использовать. Но в данной ситуации проблемами могут стать закупка данных карт из-за рубежа и аппаратная поддержка, т.к. данной карте требуется PCIe x1 и Core i5 6-го поколения для полноценной работы

Для захвата и кодировки видео будет использоваться ffmpeg и встроенный в RK3588 аппаратный кодер/декодер видео (VPU) с использованием ПО nyanmisaka/ffmpeg-rockchip. Это позволит значительно сократить потребление CPU и памяти во время работы с видеопотоком и ускорить сам процесс кодировки. Также, при дальнейшей разработке, с помощью предоставляемого Rockchip API RKMPP можно будет гораздо глубже интегрировать пользовательское ПО с VPU, повысив тем самым эффективность работы.

"Умный" блок питания

Реализация "умного" блока питания подразумевает, что в блок питания будет встроена некая система, которая будет отслеживать различные параметры цепей блока питания. Наличие подобной системы позволит отслеживать полезную статистику (энергопотребление, пиковая нагрузка и т.д.), но что гораздо более важно, поможет обнаруживать внештатные ситуации в системе питания, которые могут привести к некорректному поведению. Такими ситуациями могут быть:

- Пониженное входное напряжение (brownout по входу) (проблемы с АКБ/генератором, может привести к нестабильной работе БП и некорректным выходным напряжениям)
- Нестабильное входное напряжение (Проблемы с генератором/нагрузкой, может привести к повреждению БП и скачкам напряжения на выходе)
- Повышенное выходное напряжение (Возникает при повреждении блока питания, может привести к повреждению всей остальной системы)
- Пониженное выходное напряжение (Проблемы с блоком питания, может привести к некорректной работе всей системы)
- Аномально высокое потребление по выходной цепи/одной из линий (Повреждение одного/нескольких модулей системы)
- Аномально низкое потребление по выходной цепи/одной из линий (Некорректный запуск/работа устройства, повреждение ПО/АО)
- Температура, выходящая за рабочий диапазон (Может указывать на повреждение БП/экстремальные условия эксплуатации, требует обогрева/отключения с цель предотвращения деградации/взрыва компонентов)

Все эти, и ряд других ситуаций можно выявлять и контролировать при наличии системы мониторинга, встроенной в блок питания. Но для полноценной реализации все возможностей подобного решения оптимальным вариантом будет наличие высокоуровневой коммуникации между ЦВМ/мультиконтроллером, с целью получения точных данных о состоянии системы и выявление кросс-модульных внештатных ситуаций. (Пример: повышенное потребление + карта захвата видео не отвечает = критическое повреждение модуля захвата видео → отключение питания к модулю, запись в логи, отправка ошибки в ЦОД)

Резервное питание для сохранения in-flight данных

Во время эксплуатации устройства не исключены ситуации, когда питание будет отключено во время работы устройства. Помимо прочего, это является проблемой из-за вероятного повреждения всех файлов, с которыми велась работа в момент

потери питания, а также увеличения риска повреждения файлов, которые уже находились в ПЗУ.

В системе есть две группы ситуаций, которые приведут к подобному исходу

Потеря питания диском

Данная ситуация может возникнуть как при потери питания всей системой, так и при внезапном отключении самого диска. Второе является гораздо большей и крайне вероятной проблемой, т.к. система подразумевает возможность горячего отключения дисков, в ходе которого потеря питания не может быть решена на уровне системы и требует соответствующего АО на уровне самого диска.

Следовательно, данная проблема решается выбором индустриального SSD с наличием функциональности безопасного завершения работы после отключения питания (Power Loss Protection/PLL), которая зачастую реализуется за счёт установки ионисторов (суперконденсаторов) в цепь питание и добавление соответствующего функционала в ПО диска.

Пример такого SSD

Потеря питания всей системой

Учитывая специфику эксплуатации устройства, данная ситуация является достаточно маловероятной, но не менее неблагоприятной, т.к. может привести как к повреждению ПЗУ для хранения записи видео с камеры, так и к повреждению ПЗУ ЦВМ, что в свою очередь может привести к критическому нарушению работы системы в результате повреждения системных файлов. И если повреждения файловой системы будут достаточно серьёзными, то устройство не сможет запустить даже какую-либо систему самовосстановления.

По этим причинам наилучшим решениям будет реализация системы резервного питания, чтобы у устройства было время на безопасное завершение работы. На данный момент существует несколько способов реализовать подобную систему, но с учётом необходимости выдерживать широкий диапазон температур любые решения на основе лития отпадают, т.к. стремительно теряют показатели токоотдачи при температурах ниже нуля.

Оптимальным решением для источника резервного питания в данном случае является банк суперконденсаторов. Существует несколько технологий их реализации, но оптимальной и единственной подходящей нам по температурному диапазону (до -40) является двухслойный электролитический конденсатор.

Данная технология обладает достаточно низкой удельной ёмкостью (~0,00018 А/ч на мм^3 при напряжении 2.7В), что означает необходимость создания достаточно большого банка таких элементов чтобы обеспечить достаточное для безопасного отключения времени работы.

Данный банк ионисторов можно подключить к модулю блока питания, чтобы задействовать уже имеющиеся цепи преобразования и питания компонентов. Это

будет менее эффективно с точки зрения КПД питания, но позволит упростить схемотехнику других модулей, сократить время на переключение между источниками, а также позволит использовать уже имеющуюся систему мониторинга “умного” блока питания.