Аппаратное обеспечение IoT/CPS
Лекция 8

А. А. Подшивалов
apodshivalov@miem.hse.ru

Память микроконтроллера

Несколько общих слов

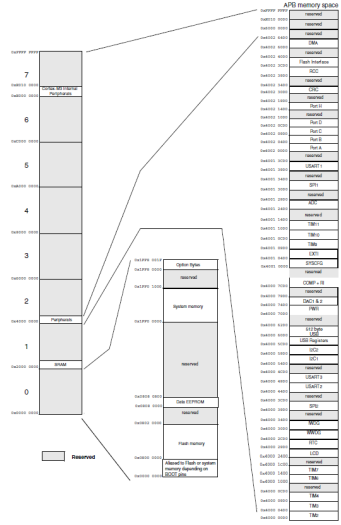
- Гарвардская архитектура — раздельная память программ и данных
- Фон-неймановская архитектура — общая память программ и данных
- Ввод-вывод с отображением на память (memory-mapped I/O)

Организация памяти МК

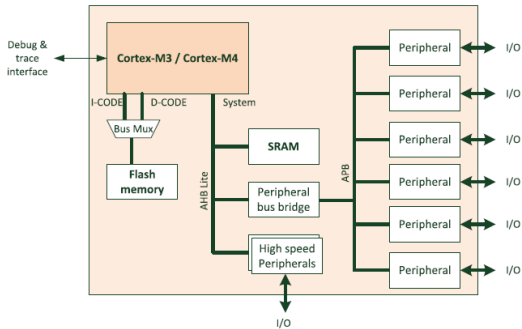
- PIC10/12/16/18
  - Типичный пример гарвардской архитектуры
  - Раздельные ROM и RAM
  - Special Function Registers в адресном пространстве RAM
- 8051
  - IRAM — внутренная RAM, 128 или 256 байт, 8-битная адресация
  - SFR в адресном пространстве IRAM
  - XRAM — до 64 кБайт (16-битная адресация)
  - PMEM — память программ, до 64 кБайт (или больше с переключаемыми страницами)

ARM Cortex-M

- Общий размер адресного пространства у 32-битного ядра — 4 Гб
- В единое адресное пространство проецируются:
  - ROM (встроенная Flash-память)
  - RAM
  - Регистры периферии
- С точки зрения программиста — типичнейший пример фон-неймановской архитектуры



ARM Cortex-M

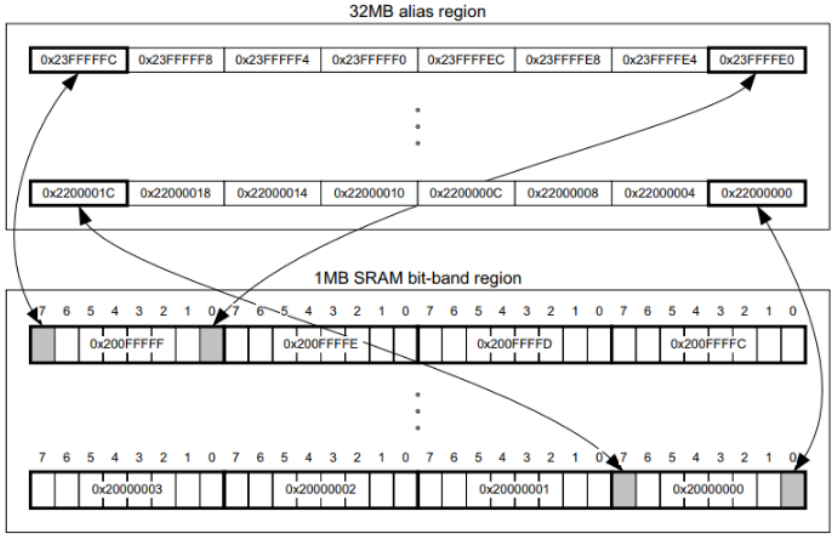


- «Внутри» ядро Cortex-M имеет гарвардскую архитектуру памяти
- Это позволяет одновременно с загрузкой инструкции из ROM загружать данные из RAM
- Выполнение кода из RAM возможно с замедлением примерно в 2 раза
- Для встроенной периферии используются шины AHB (AMBA High-performance Bus) или APB (AMBA Peripheral Bus), описанные в стандарте AMBA (Advanced Microcontroller Bus Architecture)

DMA — контроллер прямого доступа к памяти

- «Асинхронный `memcpy`»
- Может копировать данные с одного адреса в памяти на другой без участия процессорного ядра
- Автоинкремент адресов источника и приемника (настраивается)
- Может управляться от периферии (UART, SPI, I^2C, ADC, ...)
- Счетчик количества операций
- При достижении середины и конца буфера генерирует прерывание

ARM Cortex-M: bit-banding



Виды памяти

Встроенная память МК

- RAM
  - Static RAM — дорогая (6 транзисторов/бит), но не требует «регенерации» и имеет низкое энергопотребление
- ROM
  - Flash
  - EEPROM

Flash-память

- Чтение происходит «строками» по 32, 64 или 128 бит, довольно медленное
  - Контролер памяти скрывает это от программиста
  - Между ядром процессора и flash часто находится «ускоритель flash» — небольшой объем кеш-памяти
- Возможна только запись 0 на место 1
  - Стирание памяти (запись 1 во все биты) производится «страницами», типичный размер — от 256 байт до 64 кБайт
  - Ресурс выражается в циклах стирание-запись, обычно около 10 000 циклов
- Контролер flash-памяти — отдельное периферийное устройство, позволяет стирать flash и записывать туда данные
- Типичный объем — десятки-сотни кБайт (256 кБ в STM32L151CC)

EEPROM

- Возможно побайтовое чтение и запись
- Существенно больший ресурс, чем у flash, 100 000 – 1 000 000 циклов
- Типичный объем — единицы кБайт (8 кБ в STM32L151CC)

FSMC — Flexible Static Memory Controller

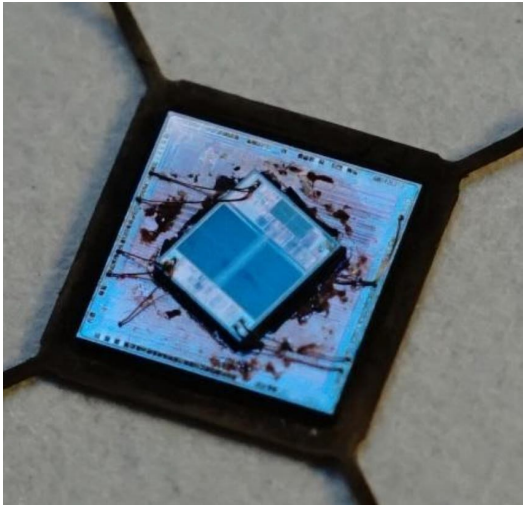
- Внешняя память проецируется в адресное пространство микроконтроллера (адреса с 0x60000000 по 0x9FFFFFFF)
- Шина адреса до 26 бит
- Шина данных до 16 бит
- «Служебные» сигналы read strobe, write strobe, chip select
- Можно подключать не только память, но и все, имеющее схожий интерфейс (например, ЖК-дисплеи с параллельной шиной)

Экзотика

- TI CC3200
  - Встроенная флеш-память очень мала и программисту недоступна
  - Программа хранится во внешней микросхеме памяти с последовательным интерфейсом QSPI, при запуске МК считывается оттуда в оперативную память
- TI CC3220SF
  - 1 Мб встроенной флеш-памяти, XIP (eXecute In Place)
  - Ускоритель не работает, память очень медленная, производительность в 1,5 раза ниже предыдущей модели
  - Внешняя микросхема памяти все равно нужна

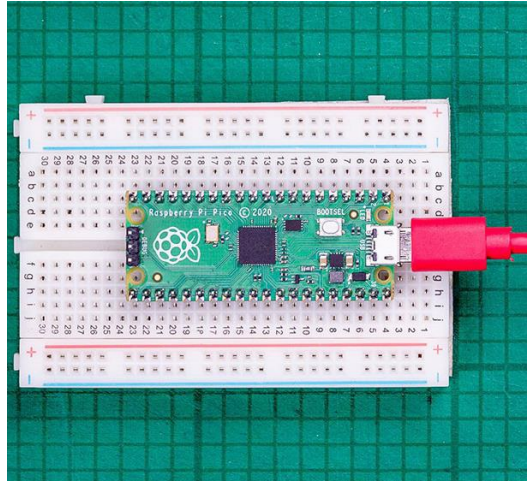
Экзотика-2

- GigaDevice — китайский производитель микросхем флеш-памяти и микроконтроллеров
- GD32 — клоны STM32 на ядре Cortex-M
- GD32V — собственные разработки на базе RISC-V
- В одном корпусе собраны микроконтроллер с периферией и отдельный чип flash-памяти с последовательным интерфейсом



Экзотика-3

- RP2040 — попытка Raspberry Pi Foundation сделать микроконтроллер
- Встроенная flash-память отсутствует, программа загружается в SRAM из внешней микросхемы памяти с интерфейсом QSPI



С точки зрения программиста

*.ld и *.map

- Linker script — файл с расширением `ld`
  - Описывает расположение данных в памяти
  - Обычно секция `.text` и копия `.data` помещаются в ROM
  - Задача обработчика сброса — обнулить `.bss` и скопировать `.data` в RAM
- Map file — файл с расширением `map`

Bootloader

- Выполняющаяся на микроконтроллере программа для записи «прошивки» в собственный Flash
- Очень часто bootloader «зашит» в микроконтроллер уже при изготовлении, и выбор программы для запуска (bootloader либо штатная прошивка) осуществляется переключением логических уровней на нескольких специально выделенных выводах
- Bootloader в STM32 умеет работать с UART и USB (по протоколу DFU, Device Firmware Update)

Постоянная память и RIOT OS

- Драйверы Flash и EEPROM
- Поддержка bootloader
- Имитация EEPROM в Flash-памяти (для некоторых семейств процессоров)
- В EEPROM/Flash можно сохранять настройки устройства, сессионные ключи, идентификаторы, ...

`drivers/include/periph/eeprom.h`

Файловые системы для Flash

- Когда flash-памяти много, можно организовать в неиспользуемой ее части полноценную файловую систему
- Основная проблема — ограниченное количество циклов перезаписи
- Решение — стирать данные только при «переполнении» ФС, поддерживать версии файлов

Математика и цифровая обработка сигналов

Практическая задача: фитнесс-трекер

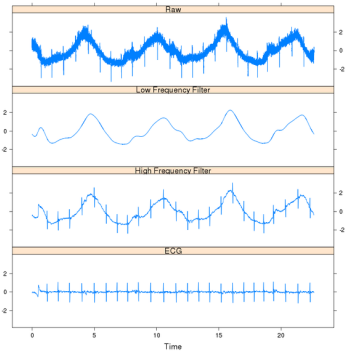
- Оцифровка ЭКГ — 16 бит, 250 отсчетов/с
- Связь — Bluetooth Low Energy
- Передаваемые данные — 10 байт/с (ЧСС, RR-интервалы)

Практическая задача: фитнесс-трекер

- Оцифровка ЭКГ — 16 бит, 250 отсчетов/с
- Связь — Bluetooth Low Energy
- Передаваемые данные — 10 байт/с (ЧСС, RR-интервалы)
- Возможность сократить передаваемые данные с 500 байт до 10 — это не только экономия энергии, но и техническая реализуемость вашего проекта

Цифровая обработка сигналов

- Преобразование Фурье
  - Дискретное преобразование Фурье, алгоритм Кули-Тьюки
- КИХ-фильтры (конечная импульсная характеристика, FIR, Finite Impulse Response)
  - Произвольная частотная характеристика
  - Большая вычислительная сложность и задержка
- БИХ-фильтры (бесконечная импульсная характеристика, IIR, Infinite Impulse Response)
  - Малая вычислительная сложность и задержка
  - Численная неустойчивость
- ...и другие подобные алгоритмы



Практические вопросы реализации

Что есть в Cortex-M?

- Cortex-M0/M1 — базовый набор команд, 32-битный умножитель
- Cortex-M3 — добавляются инструкции типа multiply-accumulate (MAC), 32-битное деление (2–12 тактов)
- Cortex-M4 — SIMD-инструкции, арифметика с насыщением
  - Опционально, в Cortex-M4F — арифметика с плавающей запятой, обычной точности (IEEE754 single precision)
- Cortex-M7 — в опциональном модуле арифметики с плавающей запятой добавляется повышенная точность (double precision)

Пример: multiply-accumulate

```
int64_t dot_product (int32_t *x, int32_t *y, int32_t N) {
    int32_t xx, yy, k;
    int64_t sum = 0;
    for(k = 0; k < N; k++) {
        xx = *x++;
        yy = *y++;
        sum += xx * yy;
    }
    return sum;
}
```

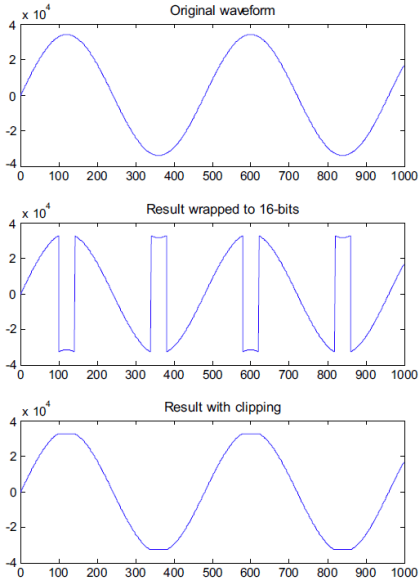
На Cortex-M3

```
int64_t dot_product (int32_t *x, int32_t *y, int32_t N) {
    int32_t xx, yy, k;
    int64_t sum = 0;
    for(k = 0; k < N; k++) {  // цикл [3 такта]
        xx = *x++;  // [2 такта]
        yy = *y++;  // [2 такта]
        sum += xx * yy;  // [от 3 до 7 тактов]
    }
    return sum;
}
```

На Cortex-M4

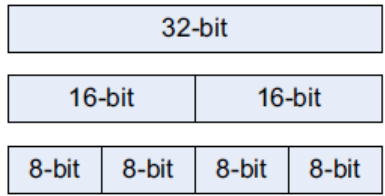
```
int64_t dot_product (int32_t *x, int32_t *y, int32_t N) {
    int32_t xx, yy, k;
    int64_t sum = 0;
    for(k = 0; k < N; k++) {  // цикл [3 такта]
        xx = *x++;  // [2 такта]
        yy = *y++;  // [1 такт, load после load]
        sum += xx * yy;  // [MAC, 1 такт]
    }
    return sum;
}
```

Арифметика с насыщением



SIMD-инструкции

- Single Instruction, Multiple Data
- Оперируем регистрами, в которые загружено несколько значений
- Обычная арифметика — сложение, умножение, multiply-accumulate

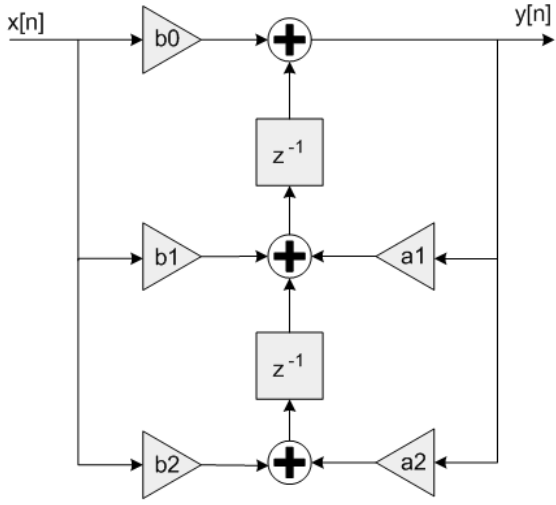


Форматы чисел с фиксированной запятой (q15, q31)

- Способ представления чисел в диапазоне от -1 до $1 - \frac{1}{2^N}$ (иногда пишут $[-1, 1)$)
- Арифметика полностью сводится к операциям с целыми числами и сдвигам
- Надо держать в голове размерность величин
- Ограниченная точность в сравнении с числами с плавающей запятой, обнуление малых значений
- bit-exactness, независимость от архитектуры

Пример: БИХ-фильтр

- $y_n = b_0 x_n + (b_1 x_{n-1} + a_1 y_{n-1}) + \\ +(b_2 x_{n-2} + a_2 y_{n-2})$
- 5 multiply-accumulate и 3 переменных состояния



Пример: БИХ-фильтр (для float)

```
// b0, b1, b2, a1, a2 - коэффициенты фильтра
// stateA, stateB, и stateC - состояние фильтра
for (sample = 0; sample < blockSize; sample++) {
    stateA = *inPtr++ + a1*stateB + a2*stateC;
    *outPtr++ = b0*stateA + b1*stateB + b2*stateC;
    stateC = stateB;
    stateB = stateA;
}
```

Как выглядит фильтр на уровне инструкций?

```
stateA = *inPtr++;     // Загрузка в регистр [2 такта]
stateA += a1*stateB;   // MAC [4 такта]
stateA += a2*stateC;   // MAC [4 такта]
out = b0*stateA;       // Умножение [2 такта]
out += b1*stateB;      // MAC [4 такта]
out += b2*stateC;      // MAC [4 такта]
*outPtr++ = out;       // Сохранение в память [2 такта]
stateC = stateB;       // Перемещение между регистрами [1 такт]
stateB = stateA;       // Перемещение между регистрами [1 такт]
                       // Цикл [3 такта]
                       // Итого 27 тактов
```

Следующий шаг

```
stateA = *inPtr++      // Загрузка в регистр [2 такта]
prod1 = a1*stateB;     // Умножение [1 такт]
prod2 = a2*stateC;     // Умножение [1 такт]
stateA += prod1;       // Сложение [1 такт]
prod4 = b1*stateB;     // Умножение [1 такт]
stateA += prod2;       // Сложение [1 такт]
out = b2*stateC;       // Умножение [1 такт]
prod3 = b0*stateA      // Умножение [1 такт]
out += prod4;          // Сложение [1 такт]
out += prod3;          // Сложение [1 такт]
stateC = stateB;       // Перемещение между регистрами [1 такт]
stateB = stateA;       // Перемещение между регистрами [1 такт]
*outPtr++ = out;       // Сохранение в память [2 такта]
                       // Цикл [3 такта]
                       // Итого 18 тактов
```

Дальнейшая оптимизация

- Loop unrolling: пишем подряд несколько (тут достаточно 3) копий тела цикла
- Переиспользуем переменные (точнее, регистры) для `stateA`, `stateB`, `stateC` в таком порядке:
  - ABC
  - CAB
  - BCA
- Загрузка данных происходит перед «развернутым» циклом
- Это сократит выполнение трех итераций с $18 \times 3 = 54$ тактов до 38 (12,67 тактов на отсчет)

CMSIS-DSP

Библиотека CMSIS-DSP

- Оптимизированная (пример выше взят оттуда) реализация для ARM Cortex-M алгоритмов ЦОС и математических функций «общего назначения»
  - Тригонометрические функции
  - Операции с матрицами и векторами
  - БИХ- и КИХ-фильтры
  - Преобразование Фурье
- Большинство функций реализовано для 32-битных `float`, `q15` и `q31`

Пример: БИХ-фильтр

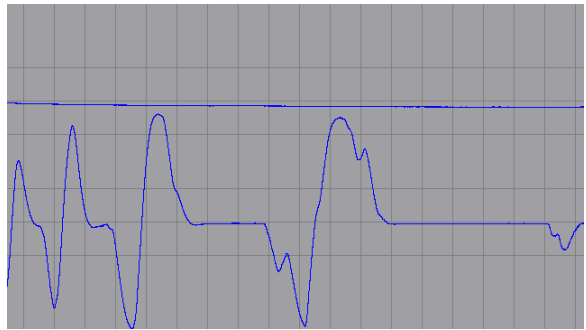
- Данные от АЦП накапливаются в буфере
- При заполнении буфера данные начинают писаться во второй буфер, а в первом фильтруются

Пример: БИХ-фильтр

```
const int num_samples = 500;
q31_t in_buf[num_samples];
q31_t out_buf[num_samples];

q31_t filter_coeffs[STAGES*5]; /* 5 coefficients for each stage */
q31_t filter_state[STAGES*4]; /* 4 state variables for each stage */
arm_biquad_cas_df1_32x32_ins_q31 filter;

arm_biquad_cas_df1_32x32_init_q31(&filter,
                                  STAGES,
                                  filter_coeffs,
                                  filter_state,
                                  0);

arm_biquad_cas_df1_32x32_q31(&filter,
                             in_buf,
                             out_buf,
                             num_samples);
```

Пример: БИХ-фильтр

- 32-битный фильтр с 32-битными переменными состояния
- ФВЧ Чебышёва 2 порядка
- Частота оцифровки — 1000 Гц, частота среза 0,1 Гц



Пример: БИХ-фильтр (64-битные переменные состояния)

```
const int num_samples = 500;
q31_t in_buf[num_samples];
q31_t out_buf[num_samples];

q31_t filter_coeffs[STAGES*5]; /* 5 coefficients for each stage */
q63_t filter_state[STAGES*4]; /* 4 state variables for each stage */
arm_biquad_cas_df1_32x64_ins_q31 filter;

arm_biquad_cas_df1_32x64_init_q31(&filter,
                                  STAGES,
                                  filter_coeffs,
                                  filter_state,
                                  0);

arm_biquad_cas_df1_32x64_q31(&filter,
                             in_buf,
                             out_buf,
                             num_samples);
```

Немного модного Machine Learning

- В CMSIS-DSP реализован SVM-классификатор
- CMSIS-NN — отдельная библиотека для вычислений с использованием сверточных нейросетей
  - Нет возможности обучения, нейросеть готовится отдельно, а затем полученные коэффициенты конвертируются в понятный CMSIS-NN формат

CMSIS-NN — производительность

- Задача классификации изображений из датасета CIFAR-10 (32×32)
- Сверточная сеть из 7 слоев, работает на Cortex-M7@216 МГц
- Около 10 изображений/с