Содержание

Отзывы на книгу «Цифровая схемотехника и архитектура компьютера. RISC-V» ... 13
Об авторах ... 14
Предисловие к русскому изданию ... 15
Предисловие от редактора русского перевода ... 22
Предисловие ... 24

Глава 1 От нуля до единицы 31

1.1. План игры ... 31
1.2. Искусство управления сложностью ... 32
1.2.1. Абстракция ... 33
1.2.2. Конструкторская дисциплина ... 35
1.2.3. Три базовых принципа ... 36
1.3. Цифровая абстракция ... 38
1.4. Системы счисления ... 40
1.4.1. Десятичная система счисления ... 40
1.4.2. Двоичная система счисления ... 41
1.4.3. Шестнадцатеричная система счисления ... 43
1.4.4. Байт, полубайт и «весь этот джаз» ... 45
1.4.5. Сложение двоичных чисел ... 46
1.4.6. Знак двоичных чисел ... 47
1.5. Логические элементы ... 53
1.5.1. Логический элемент НЕ ... 53
1.5.2. Буфер ... 54
1.5.3. Логический элемент И ... 54
1.5.4. Логический элемент ИЛИ ... 54
1.5.5. Другие логические элементы с двумя входными сигналами ... 55
1.5.6. Логические элементы с количеством входов больше двух ... 56
1.6. За пределами цифровой абстракции ... 57
1.6.1. Напряжение питания ... 57
1.6.2. Логические уровни ... 57
1.6.3. Допускаемые уровни шумов ... 58
1.6.4. Передаточная характеристика ... 59
1.6.5. Статическая дисциплина ... 60
1.7. КМОП-транзисторы ... 62
1.7.1. Полупроводники ... 63
1.7.2. Диоды ... 64
1.7.3. Конденсаторы ... 64
1.7.4. *n*-МОП- и *p*-МОП-транзисторы ... 65
1.7.5. Логический элемент НЕ на КМОП-транзисторах ... 69
1.7.6. Другие логические элементы на КМОП-транзисторах ... 69
1.7.7. Передаточный логический элемент ... 72
1.7.8. Псевдо-*n*-МОП-логика ... 72
1.8. Потребляемая мощность ... 73
1.9. Краткий обзор главы 1 и того, что нас ждет впереди ... 75
Упражнения ... 77
Вопросы для собеседования ... 89

Глава 2 Разработка комбинационной логики 91

2.1. Введение ... 91
2.2. Логические функции ... 95
2.2.1. Терминология ... 95
2.2.2. Дизъюнктивная форма ... 96
2.2.3. Конъюнктивная форма ... 98
2.3. Булева алгебра ... 99
2.3.1. Аксиомы ... 100
2.3.2. Теоремы одной переменной ... 100
2.3.3. Теоремы с несколькими переменными ... 102
2.3.4. Доказательство теорем булевой алгебры ... 104
2.3.5. Упрощение логических уравнений ... 105
2.4. От логики к логическим элементам ... 106
2.5. Многоуровневая комбинационная логика ... 110
2.5.1. Минимизация аппаратных затрат ... 111
2.5.2. Перемещение инверсии ... 112
2.6. Что такое X и Z? ... 115
2.6.1. Недопустимое значение: X ... 115
2.6.2. Третье состояние: Z ... 116
2.7. Карты Карно ... 118
2.7.1. Думайте об овалах ... 119
2.7.2. Логическая минимизация на картах Карно ... 120
2.7.3. Безразличные переменные ... 124
2.7.4. Карты Карно: подведение итогов ... 124
2.8. Базовые комбинационные блоки ... 125
2.8.1. Мультиплексоры ... 125
2.8.2. Дешифраторы ... 129
2.9. Временные характеристики ... 131
2.9.1. Задержка распространения и задержка реакции ... 131
2.9.2. Импульсные помехи ... 136
2.10. Заключение ... 139
Упражнения ... 140
Вопросы для собеседования ... 147

Глава 3 Разработка последовательностной логики 149

3.1. Введение ... 149
3.2. Защелки и триггеры ... 150
3.2.1. RS-триггер ... 151
3.2.2. D-защелка ... 154
3.2.3. D-триггер ... 155
3.2.4. Регистр ... 156
3.2.5. Триггер с функцией разрешения ... 156
3.2.6. Триггер с функцией сброса ... 158
3.2.7. Разработка триггеров и защелок на транзисторном уровне ... 159
3.2.8. Сравнение защелок и триггеров ... 160
3.3. Разработка синхронных логических схем ... 161
3.3.1. Некоторые проблемные схемы ... 161
3.3.2. Синхронные последовательностные схемы ... 163
3.3.3. Синхронные и асинхронные схемы ... 166
3.4. Конечные автоматы ... 166
3.4.1. Пример разработки конечного автомата ... 167
3.4.2. Кодирование состояний ... 173

3.4.3. Автоматы Мура и Мили ... 176
3.4.4. Декомпозиция конечных автоматов ... 180
3.4.5. Восстановление конечных автоматов по электрической схеме ... 182
3.4.6. Конечные автоматы: подведение итогов ... 185
3.5. Синхронизация последовательностных схем ... 185
3.5.1. Динамическая дисциплина ... 187
3.5.2. Временные характеристики системы ... 188
3.5.3. Расфазировка тактовых сигналов ... 194
3.5.4. Метастабильность ... 197
3.5.5. Синхронизаторы ... 199
3.5.6. Вычисление времени разрешения ... 201
3.6. Параллелизм ... 205
3.7. Заключение ... 209
Упражнения ... 210
Вопросы для собеседования ... 218

Глава 4 Языки описания аппаратуры 221

4.1. Введение ... 221
4.1.1. Модули ... 222
4.1.2. Происхождение языков SystemVerilog и VHDL ... 222
4.1.3. Моделирование и синтез ... 224
4.2. Комбинационная логика ... 226
4.2.1. Битовые операторы ... 227
4.2.2. Комментарии и пробелы ... 229
4.2.3. Операторы сокращения ... 230
4.2.4. Условное присваивание ... 230
4.2.5. Внутренние переменные ... 233
4.2.6. Приоритет ... 235
4.2.7. Числа ... 235
4.2.8. Z-состояние и X-состояние ... 237
4.2.9. Манипуляция с битами ... 239
4.2.10. Задержки ... 239
4.3. Структурное моделирование ... 241
4.4. Последовательностная логика ... 245
4.4.1. Регистры ... 245
4.4.2. Регистры со сбросом ... 245
4.4.3. Регистры с сигналом разрешения ... 248
4.4.4. Группы регистров ... 249
4.4.5. Защелки ... 250
4.5. И снова комбинационная логика ... 251
4.5.1. Операторы case ... 254
4.5.2. Условный оператор (if) ... 256
4.5.3. Таблицы истинности с незначащими битами ... 259
4.5.4. Блокирующие и неблокирующие присваивания ... 260
4.6. Конечные автоматы ... 264
4.7. Типы данных ... 268
4.7.1. SystemVerilog ... 268
4.7.2. VHDL ... 269
4.8. Параметризированные модули ... 272
4.9. Тестбенч ... 275
4.10. Заключение ... 280
Упражнения ... 281
Упражнения для SystemVerilog ... 287

Упражнения для VHDL ... 289
Вопросы для собеседования ... 291

Глава 5 Цифровые функциональные узлы 293

5.1. Введение ... 293
5.2. Арифметические схемы ... 294
5.2.1. Сложение ... 294
5.2.2. Вычитание ... 302
5.2.3. Компараторы ... 303
5.2.4. Арифметико-логическое устройство ... 304
5.2.5. Схемы сдвига и циклического сдвига ... 309
5.2.6. Умножение ... 310
5.2.7. Деление ... 312
5.2.8. Дополнительная литература ... 313
5.3. Представление чисел ... 313
5.3.1. Числа с фиксированной запятой ... 314
5.3.2. Числа с плавающей запятой ... 315
5.4. Функциональные узлы последовательностной логики ... 319
5.4.1. Счетчики ... 319
5.4.2. Сдвиговые регистры ... 321
5.5. Матрицы памяти ... 324
5.5.1. Обзор матриц памяти ... 324
5.5.2. Динамическое ОЗУ (DRAM) ... 328
5.5.3. Статическое ОЗУ (SRAM) ... 328
5.5.4. Площадь и задержки ... 329
5.5.5. Регистровые файлы ... 330
5.5.6. Постоянное запоминающее устройство ... 330
5.5.7. Реализация логических функций с использованием матриц памяти ... 332
5.5.8. Языки описания аппаратуры и память ... 333
5.6. Матрицы логических элементов ... 336
5.6.1. Программируемые логические матрицы ... 336
5.6.2. Программируемые пользователем вентильные матрицы ... 338
5.6.3. Схемотехника матриц ... 345
5.7. Заключение ... 346
Упражнения ... 347
Вопросы для собеседования ... 357

Глава 6 Архитектура 359

6.1. Предисловие ... 359
6.2. Язык ассемблера ... 362
6.2.1. Инструкции ... 362
6.2.2. Операнды: регистры, память и константы ... 364
6.3. Программирование ... 370
6.3.1. Порядок выполнения программы ... 371
6.3.2. Арифметические / логические инструкции ... 371
6.3.3. Ветвление программ ... 374
6.3.4. Условные операторы ... 377
6.3.5. Циклы ... 378
6.3.6. Массив ... 381
6.3.7. Вызовы функций ... 385
6.3.8. Псевдокоманды ... 398
6.4. Машинный язык ... 400

6.4.1. Инструкции типа *R* ... *401*
6.4.2. Инструкции типа I ... 403
6.4.3. Инструкции типа *S*/*B* ... *404*
6.4.4. Инструкции типа *U*/*J* ... *407*
6.4.5. Кодирование констант ... 408
6.4.6. Режимы адресации ... 409
6.4.7. Расшифровываем машинные коды ... 411
6.4.8. Могущество хранимой программы ... 412
6.5. Камера, мотор! Компилируем, ассемблируем и загружаем ... 413
6.5.1. Карта памяти ... 414
6.5.2. Директивы ассемблера ... 416
6.5.3. Компиляция ... 419
6.5.4. Трансляция ... 421
6.5.5. Компоновка ... 424
6.5.6. Загрузка ... 426
6.6. Добавочные сведения ... 426
6.6.1. Порядок байтов ... 426
6.6.2. Исключения ... 427
6.6.3. Команды для чисел со знаком и без знака ... 431
6.6.4. Команды для работы с числами с плавающей запятой ... 433
6.6.5. Сжатые инструкции ... 434
6.7. Эволюция архитектуры RISC-V ... 436
6.7.1. Базовые наборы команд и расширения RISC-V ... 436
6.7.2. Сравнение архитектур RISC-V и MIPS ... 437
6.7.3. Сравнение архитектур RISC-V и ARM ... 438
6.8. Живой пример: архитектура x86 ... 439
6.8.1. Регистры x86 ... 440
6.8.2. Операнды x86 ... 440
6.8.3. Флаги состояния ... 442
6.8.4. Команды x86 ... 442
6.8.5. Кодировка команд x86 ... 444
6.8.6. Другие особенности x86 ... 446
6.8.7. Архитектура x86: подведение итогов ... 447
6.9. Заключение ... 448
Упражнения ... 449
Вопросы для собеседования ... 462

Глава 7 Микроархитектура 465

7.1. Введение ... 465
7.1.1. Архитектурное состояние и система команд ... 466
7.1.2. Процесс разработки ... 466
7.1.3. Микроархитектуры RISC-V ... 469
7.2. Анализ производительности ... 470
7.3. Однотактный процессор ... 472
7.3.1. Пример программы ... 473
7.3.2. Однотактный тракт данных ... 473
7.3.3. Однотактный блок управления ... 482
7.3.4. Дополнительные команды ... 485
7.3.5. Анализ производительности ... 488
7.4. Многотактный процессор ... 490
7.4.1. Многотактный тракт данных ... 491
7.4.2. Многотактное устройство управления ... 497
7.4.3. Дополнительные команды ... 509

7.4.4. Анализ производительности ... 512
7.5. Конвейерный процессор ... 515
7.5.1. Конвейерный тракт данных ... 518
7.5.2. Конвейерное устройство управления ... 520
7.5.3. Конфликты ... 520
7.5.4. Анализ производительности ... 531
7.6. Разрабатываем процессор на HDL ... 533
7.6.1. Однотактный процессор ... 535
7.6.2. Универсальные строительные блоки ... 539
7.6.3. Тестбенч ... 542
7.7. Улучшенные микроархитектуры ... 547
7.7.1. Длинные конвейеры ... 548
7.7.2. Микрокоманды ... 549
7.7.3. Предсказание условных переходов ... 550
7.7.4. Суперскалярный процессор ... 552
7.7.5. Процессор с внеочередным выполнением команд ... 555
7.7.6. Переименование регистров ... 558
7.7.7. Многопоточность ... 560
7.7.8. Мультипроцессоры ... 561
7.8. Живой пример: эволюция микроархитектуры RISC-V ... 565
7.9. Заключение ... 569
Упражнения ... 571
Вопросы для собеседования ... 579

Глава 8 Системы памяти 581

8.1. Введение ... 581
8.2. Анализ производительности систем памяти ... 586
8.3. Кеш-память ... 588
8.3.1. Какие данные хранятся в кеш-памяти? ... 589
8.3.2. Как найти данные в кеш-памяти? ... 590
8.3.3. Какие данные заместить в кеш-памяти? ... 599
8.3.4. Улучшенная кеш-память ... 600
8.4. Виртуальная память ... 604
8.4.1. Трансляция адресов ... 607
8.4.2. Таблица страниц ... 609
8.4.4. Защита памяти ... 612
8.4.5. Стратегии замещения страниц ... 612
8.4.6. Многоуровневые таблицы страниц ... 613
8.5. Заключение ... 616
Эпилог ... 616
Упражнения ... 617
Вопросы для собеседования ... 624

Глава 9 Ввод/вывод во встраиваемых системах 626

9.1. Введение ... 626
9.2. Отображение ввода/вывода в пространство памяти ... 627
9.3. Ввод/вывод во встраиваемых системах ... 629
9.3.1. Плата RED-V ... 629
9.3.2. Система на кристалле FE310-G002 ... 631
9.3.3. Цифровой ввод/вывод общего назначения ... 634
9.3.4. Драйверы устройств ввода/вывода ... 638
9.3.5. Последовательный ввод/вывод ... 642

9.3.6. Таймеры ... 659
9.3.7. Аналоговый ввод/вывод ... 661
9.3.8. Прерывания ... 669
9.4. Другие внешние устройства микроконтроллера ... 674
9.4.1. Символьные ЖК-дисплеи ... 674
9.4.2. VGA-монитор ... 678
9.4.3. Беспроводная связь Bluetooth ... 684
9.4.4. Управление двигателями ... 686
9.5. Заключение ... 698

Приложение A. Реализация цифровых систем 699

A.1. Введение ... 699
A.2. Логические микросхемы серии 74xx ... 700
A.2.1. Логические элементы ... 700
A.2.2. Другие логические функции ... 701
A.3. Программируемая логика ... 703
A.3.1. PROM ... 704
A.3.2. Блоки PLA ... 705
A.3.3. FPGA ... 705
A.4. Заказные специализированные интегральные схемы ... 708
A.5. Работа с документацией ... 709
A.6. Семейства логических микросхем ... 714
A.7. Корпуса и монтаж интегральных схем ... 717
A.8. Линии передачи ... 721
A.8.1. Согласованная нагрузка ... 723
A.8.2. Нагрузка холостого хода ... 725
A.8.3. Нагрузка короткого замыкания ... 726
A.8.4. Рассогласованная нагрузка ... 726
A.8.5. Когда нужно применять модели линии передачи ... 729
A.8.6. Правильное подключение нагрузки к линии передачи ... 730
A.8.7. Вывод формулы для Z_0 ... 731
A.8.8. Вывод формулы для коэффициента отражения ... 733
A.8.9. Линии передачи: подведение итогов ... 733
A.9. Экономика ... 735

Приложение B. Система команд RISC-V 738

Приложение C. Программирование на языке C 747

C.1. Введение ... 747
Краткий итог ... 749
C.2. Добро пожаловать в язык C ... 750
C.2.1. Структура программы на языке C ... 750
C.2.2. Запуск C-программы ... 751
Краткий итог ... 752
C.3. Компиляция ... 752
C.3.1. Комментарии ... 753
C.3.2. `#define` ... 753
C.3.3. `#include` ... 754
Краткий итог ... 755
C.4. Переменные ... 756
C.4.1. Базовые типы данных ... 756
C.4.2. Глобальные и локальные переменные ... 758

С.4.3. Инициализация переменных ... 759
Краткий итог ... 759
С.5. Операции ... 760
С.6. Вызовы функций ... 763
С.7. Управление последовательностью выполнения действий ... 765
С.7.1. Условные операторы ... 765
С.7.2. Циклы ... 767
Краткий итог ... 769
С.8. Другие типы данных ... 770
С.8.1. Указатели ... 770
С.8.2. Массивы ... 772
С.8.3. Символы ... 777
С.8.4. Строки символов ... 778
С.8.5. Структуры ... 780
С.8.6. Оператор `typedef` ... 781
С.8.7. Динамическое распределение памяти ... 783
С.8.8. Связные списки ... 784
Краткий итог ... 786
С.9. Стандартная библиотека языка С ... 786
С.9.1. `stdio` ... 787
С.9.2. `stdlib` ... 791
С.9.3. `math` ... 794
С.9.4. `string` ... 794
С.10. Компилятор и опции командной строки ... 795
С.10.1. Компиляция нескольких исходных с-файлов ... 795
С.10.2. Опции компилятора ... 795
С.10.3. Аргументы командной строки ... 796
С.11. Типичные ошибки ... 797

Дополнительная литература 801

Предметный указатель 803

Отзывы на книгу «Цифровая схемотехника и архитектура компьютера. RISC-V»

Харрис и Харрис детально описали устройство процессора RISC-V от электронных компонентов до микроархитектуры. Их ясные объяснения в сочетании с широким охватом темы дают полное представление как о цифровой схемотехнике, так и об архитектуре RISC-V. Это очень информативный и познавательный подход, поскольку у студентов есть отличная возможность запускать большие цифровые проекты на современных FPGA.

Дэвид А. Паттерсон, Калифорнийский университет в Беркли

Потрясающе, какие разнообразные знания авторы объединили в одной книге! По мере развития производства полупроводников значимость правильной разработки цифровых схем и компьютерной архитектуры будет только возрастать. Читатели найдут доступное и всестороннее рассмотрение обеих тем и после прочтения книги получат четкое понимание архитектуры набора команд RISC-V.

Эндрю Уотерман, SiFive

Мне доводилось видеть отличные учебники по цифровой схемотехнике и отличные учебники по компьютерным архитектурам – но этот учебник представляет собой и то, и другое! Он также уникален своей способностью формировать общую картину. Авторы начинают с азов, и это делает архитектуру RISC-V понятной. Упражнения к главам этой книги послужат отличным методическим ресурсом для университетских преподавателей.

Рой Кравиц, Государственный университет Портленда

Когда я впервые прочитал учебник по MIPS в 2008 году, то подумал, что это один из лучших учебников по компьютерной архитектуре. Я сразу начал использовать его в своих лекциях. Тринадцать лет спустя мне посчастливилось прочитать новое издание про RISC-V, и мое мнение осталось прежним: это отличная книга, очень понятная, исчерпывающая, с высоким образовательным потенциалом. Она полностью соответствует учебной программе, которую проходят студенты в области цифровой схемотехники и компьютерной архитектуры. Я с нетерпением жду возможности использовать этот учебник по архитектуре RISC-V в своих лекциях.

Даниэль Чавер Мартинес, Мадридский университет Комплутенсе

Об авторах

Дэвид Мани Харрис (David Money Harris) – доцент в колледже им. Харви Мадда (Harvey Mudd College). Получил ученую степень кандидата наук по электронике в Стэнфордском университете и степень магистра по электронике и информатике в Массачусетском технологическом институте (MIT). Перед Стэнфордом работал в компании Intel в качестве схемотехника и разработчика логики для процессоров Itanium и Pentium II. Впоследствии работал консультантом в Sun Microsystems, Hewlett-Packard, Evans & Sutherland и других компаниях.

Увлечения Дэвида включают в себя преподавание, разработку чипов и активный отдых на природе. В свободное от работы время занимается пешим туризмом, скалолазанием и альпинизмом. Особенно любит длинные прогулки с сыном Абрахамом, который родился, когда Дэвид начал работать над этой книгой. Дэвид имеет более десяти патентов и является автором трех других учебников по разработке чипов, а также двух путеводителей по горам Южной Калифорнии.

Сара Л. Харрис (Sarah L. Harris) – доцент в колледже им. Харви Мадда (Harvey Mudd College). Получила степени магистра и кандидата наук по электронике в Стэнфордском университете и степень бакалавра по электронике и вычислительной технике в университете Брайама Янга (BrighamYoung University). Сара также работала в компаниях Hewlett-Packard, San Diego Supercomputer Center, Nvidia и исследовательском отделе компании Microsoft Research в Пекине.

Интересы Сары не ограничиваются преподаванием, изучением и разработкой новых технологий, она также любит путешествовать, увлекается виндсерфингом, скалолазанием и игрой на гитаре. Среди ее недавних начинаний можно отметить исследования в области интерфейсов, позволяющих разрабатывать цифровые электрические схемы простыми рисунками от руки, работу в качестве научного корреспондента для филиала Национального общественного радио (National Public Radio) и обучение кайтсерфингу. Сара говорит на четырех языках и собирается изучить еще несколько в ближайшем будущем.

Романов Александр Юрьевич – научный редактор русского перевода данной книги, доцент Московского института электроники и математики им. А. Н. Тихонова Национального исследовательского университета «Высшая школа экономики» (МИЭМ НИУ ВШЭ). В 2009 г. закончил магистратуру в Харьковском политехническом институте, работал в Киевском политехническом институте им. Сикорского. С 2014 г. работает в МИЭМ НИУ ВШЭ, где возглавляет лабораторию САПР (https://miem.hse.ru/edu/ce/cadsystem), специализирующуюся на проектной деятельности, а также разработке цифровых систем на ПЛИС/микроконтроллерах, робототехнических комплексов, аппаратных реализаций систем искусственного интеллекта, многопроцессорных систем, систем удаленного доступа к лабораторному оборудованию и т. д. В 2015 г. защитил диссертацию в Институте проблем проектирования в микроэлектронике РАН (г. Зеленоград), является автором более 150 научных статей, патентов и книг. Более подробно об учебном процессе в лаборатории можно узнать из интервью: https://miem.hse.ru/news/364316102.html.

Предисловие к русскому изданию

Вы держите в руках книгу, которая занимает на российском книжном рынке особое место. Если вы студент и хотите пройти собеседование в крупную электронную компанию на прозицию проектировщика процессоров, нейроускорителей или сетевых микросхем, то самое лучшее, что вы можете сделать сейчас, – это прочитать данную книгу от корки до корки, одновременно выполняя упражнения на симуляторах и платах ПЛИС.

Когда мы говорим о собеседованиях, мы говорим о таких компаниях, как Apple, Intel, NVidia, а также о передовых российских проектировщиках процессоров Syntacore, «Элвис-НеоТек» и «Байкал Электроникс». В каждой из них вам дадут задания типа «напишите на доске дизайн простого арбитра на языке описания аппаратуры Verilog» или «объясните, как помогают производительности микропроцессора байпасы в его конвейере».

Конечно, мы не утверждаем, что изучение этого учебника гарантирует вам успех, но эта книга закладывает современную базу во всех областях, о которых вас будут спрашивать: цифровая логика и ее тайминг, арифметические блоки и конечные автоматы, архитектура (система команд) и микроархитектура (строение конвейера) процессора. С использованием того же самого языка SystemVerilog, который используют современные разработчики цифровых систем на рабочем месте (вам также могут встретиться блоки на языке VHDL, и он тоже есть в книге).

После этой книги вам, конечно, нужно будет сделать несколько учебных проектов и изучить по статьям в сети Интернет некоторые типы дизайнов, которых в книге нет (очереди FIFO, пересечение доменов тактовой частоты и т. д.). Совместно с этой книгой также рекомендуется читать еще одну – «Цифровой синтез: практический курс»[1]. Она создана специально как дополнение к предыдущей версии книги Харрисов; в ближайшее время планируется ее переиздание, адаптированное под RISC-V. После этого вы будете готовы к бою. Никакая другая книга или комбинация книг на русском языке не поможет вам пройти эту начальную часть траектории эффективнее, чем «Цифровая схемотехника и архитектура компьютера: RISC-V» Дэвида Харриса и Сары Харрис.

Как возникла современная база проектирования

В 1980-е годы произошли две революции в проектировании цифровых микросхем. Первая революция была в маршруте проектирования. До конца 1980-х схемы рисо-

[1] Цифровой синтез: практический курс / под общ. ред. А. Ю. Романова, Ю. В. Панчула. М.: ДМК Пресс, 2020. https://dmkpress.com/catalog/electronics/circuit_design/978-5-97060-850-0/.

вали мышкой на экране, а с начала 1990-х их стали синтезировать из кода на языках описания аппаратуры Verilog и VHDL. Основные события:

- 1984 – Gateway Design Automation / Cadence изобретают язык описания аппаратуры Verilog;
- 1984 – Xilinx изобретает реконфигурируемые микросхемы ПЛИС/FPGA;
- 1986 – Optimal Solutions / Synopsys изобретают цифровой синтез;
- 1988–1992 – цифровой синтез внедряют в проектирование Apple, Sun, Nokia и др.;
- 1997–1999 – Lexra, MIPS, ARM начинают лицензировать процессорные ядра в виде IP-блоков (Intellectual Property – интеллектуальная собственность).

Вторая революция произошла в архитектуре и микроархитектуре процессоров. В 1970-х были популярны процессоры с двухуровневой организацией на основе так называемой технологии микропрограммирования. Команды процессора, видимые программисту, реализовывались на аппаратном уровне с помощью цепочек из слов (последовательностей битов в памяти) с сигналами контроля, так называемого микрокода. Такая организация позволяла создавать очень сложные системы команд, но ограничивала возможности по их параллельному выполнению.

В 1978 году группа исследователей в Стенфорде под руководством Джона Хеннесси задала себе вопрос: действительно ли нужны эти сложные команды, или их необходимость – просто маркетинговая иллюзия? Стенфордцы провели анализ большого количества пользовательских программ и пришли к выводу, что большинство используемых в программах команд – простые. И если тратить усилия не на усложнение цепочек микрокода, а на построение так называемого конвейера – структуры, в которой простые команды выполняются с перекрытием во времени, – то можно выполнять программы быстрее. Так появилась архитектура MIPS.

К похожим идеям пришла группа в Беркли под руководством Дэвида Паттерсона, которая в начале 1980-х создала архитектуры RISC I и RISC II, из которых выросла архитектура SPARC. В середине 1980-х появилась компания ARM, и за последующие десятилетия процессоры с новой организацией сначала завоевали рынок рабочих станций, а потом и бытовой электроники, сотовых телефонов и микроконтроллеров.

В конце 1980-х даже Intel, которая изначально делала процессоры на основе микрокода, стала вводить в Intel 486 конвейер, а к 1996 году построила процессор PentiumPro, в котором большинство команд на лету преобразовывались в простые команды, отправлявшиеся на конвейер в стиле RISC-процессоров. Хранимый в памяти микрокод остался только для сложных инструкций.

В начале 1990-х основатели концепции RISC-процессоров Джон Хеннесси и Дэвид Паттерсон опубликовали два учебника, которые стали бестеллерами:

- учебник начального уровня «Архитектура компьютера и проектирование компьютерных систем»
- и более сложный учебник «Компьютерная архитектура: количественный подход».

Эти учебники описывали архитектуру и микроархитектуру сначала на основе MIPS-образной архитектуры DLX, а потом стали использовать MIPS. К тому времени процессоры архитектуры MIPS уже использовались в компьютерах для голливудских спецэффектов, а потом и в домашней электронике.

В течение 1990-х американские университеты внедрили в учебный процесс книги Хеннесси и Паттерсона, курсы по языкам описания аппаратуры Verilog и VHDL, а также лабораторные работы на платах с микросхемами реконфигурируемой логики ПЛИС/FPGA, которые позволили строить студенческие процессоры без сложной процедуры заказа их на фабрике. Так выросло поколение студентов, которые разработали Apple iPhone, графические процессоры от NVidia, микросхемы для маршрутизаторов Cisco и Juniper и другие популярные устройства.

Что происходило в это время в России

Революции в цифровом синтезе и микроархитектуре процессоров по времени выпали на сложный период российской истории. Открытие советского рынка для иностранных компьютеров, коллапс СССР и недофинансирование вузов привели к тому, что в российском обществе перестали верить, что в России возможно проектирование конкурентоспособных чипов.

Долгое время группы разработчиков сохранялись только в компаниях, связанных с обороной и космосом, для проектирования чипов для космоса в таких организациях, как НИИСИ и НПЦ «Элвис». Российская команда, разработавшая процессор «Эльбрус», прототип которого при симуляции на Verilog показывал многообещающие результаты на вычислениях с плавающей запятой, попыталась в 2000 году получить финансирование у венчурных капиталистов Кремниевой долины, но вернулась в Россию.

В результате обучение компьютерной архитектуре во многих российских вузах стало описательным. Например, вузовские преподаватели стали использовать учебник Эндрю Таненбаума «Архитектура компьютеров», который был больше ориентирован на программистов, чем разработчиков процессоров. Что и понятно – Таненбаум получил известность как создатель операционной системы Minix, предшественницы Linux, а не разработчик процессора. Для микроархитектуры учебник использовал предыдущую технологическую базу (микрокод) и никак не был привязан к синтезу процессоров на языках описания аппаратуры. То есть студенты изучали системы команд и виды кеша для программистов, но не могли сделать процессор руками.

Учебники Паттерсона и Хеннесси были переведены на русский язык с большим опозданием, и в них не вошли приложения с описанием языков проектирования аппаратуры. Профессор Аркадий Поляков после работы в Кремниевой долине вернулся в Россию и издал в 2003 году учебник по Verilog, но в нем не было привязки к компьютерной архитектуре. Даже когда российские вузы делали лабораторные работы с ПЛИС, преподаватели часто выбирали разработку схемы с помощью рисования мышкой на экране, хотя в американских компаниях это перестали делать еще в начале 1990-х. В типичной вузовской методичке по цифровой электронике 2000-х годов шло качественное описание схем мультиплексоров и триггеров, а потом, пропустив

два уровня абстракции, студенты сразу изучали программирование микроконтроллеров. Не было учебника, который бы увязывал все эти элементы в одно целое.

История появления учебника «Цифровая схемотехника и архитектура компьютера»

Дэвид Харрис учился в MIT как раз тогда, когда произошла революция в маршруте проектирования конца 1980 – начала 1990-х годов. Вооруженный новыми методологиями, Дэвид пошел работать в Intel над процессором Pentium II. После этого защитил диссертацию в Стенфорде и стал преподавателем в Колледже Харви-Мадд в южной Калифорнии.

- http://pages.hmc.edu/harris/about/General_Resume.pdf.

Колледж Харви-Мадд не особо известен широкой публике, но находится среди топ-университетов по заработным платам выпускников, а также количеству выпускников, защищающих впоследствии диссертации. Еще Харви-Мадд известен проектами в области робототехники, которые они делают вместе с NASA. Иными словами, это практик высшего калибра.

- https://www.monster.com/career-advice/article/colleges-that-get-most-pay-for-graduates.
- https://www.hmc.edu/about-hmc/2020/09/14/harvey-mudd-ranks-high-in-u-s-news-and-world-report-2021/.
- https://ti.arc.nasa.gov/news/ASR-hosts-Clinic-project/.

Дэвид Харрис и его коллега Сара Харрис (они не родственники, а просто однофамильцы) в 2008 году написали первый вариант учебника, в котором в лаконичной и технически корректной форме изложили материал, который обычно входил в несколько учебников: цифровая логика, языки описания аппаратуры Verilog и VHDL, архитектура и микроархитектура компьютера, а также использование готовых чипов. Студенты получили возможность, используя только один учебник, начать с нуля, дойти до конструирования собственного небольшого процессора, реализующего подмножество архитектуры MIPS, а потом сравнить его работу с реальным микроконтроллером Microchip PIC32 на архитектуре MIPS.

Книга Харрисов появилась в России

В начале 2010-х годов в российской электронной индустрии наступило оживление. Зеленоградские компании «Элвис» и «Миландр» налаживали контакты с ARM и MIPS для лицензирования процессорных ядер, НИИСИ строил суперскалярное 64-битное MIPS-ядро, КМ211 разрабатывали процессоры для смарт-карт и налаживали контакты с тайваньской фабрикой TSMC. РОСНАНО финансировало проект компании «Элвис» в области умных камер и новую компанию «Байкал Электроникс».

Когда проблемы недостатка финансирования и изоляции российских компаний от международного рынка стали решаться, на первый план вышла проблема нехватки кадров. Хотя вузовские программы в МИЭТ и ИТМО старались поддерживать свои программы на уровне, компаниям приходилось обучать не только разработчиков схем на Verilog (на уровне RTL – Register Transfer Level), но и инженеров-верификаторов, которым нужно было создавать тесты и модели со знанием, что происходит в схеме, спроектированной на Verilog.

Поэтому когда в 2014 году появилась идея перевести на русский язык книгу Харрисов, ее поддержали сразу несколько человек и компаний. Преподаватели и аспиранты российских университетов МИФИ, ИТМО, ИТМиВТ, СПб ГУАП, украинских КНУ, КПИ, ХНУРЭ и ЧНТУ; сотрудники российских компаний МЦСТ, НИИСИ РАН, «Модуль», RusBITech, amperka.ru, Runtime Design Automation, «БиДжи»; русские инженеры американских и европейских компаний Imagination Technologies / MIPS Processors, AMD, Synopsys, Apple, eASIC, Cadence, NVidia, Marvell Semiconductor, университета Принстон – более 40 человек приняли участие в переводе, ревью, редактировании и корректировании как учебника, так и лекционных слайдов для него.

Перевод поддержала британская компания Imagination Technologies, которая в это время заключала сделки по лицензированию процессорных ядер MIPS и графических ядер PowerVR с российскими компаниями и была заинтересована в улучшении технического образования в России для налаживания долговременных бизнес-отношений с российскими партнерами. В издании книги также помогло eNano, образовательное отделение РОСНАНО, российского фонда, который вкладывал в микроэлектронные проекты.

После выхода первого онлайн-издания за дело взялось российское издательство «ДМК Пресс», которое выпустило второе издание Харрисов (использующее архитектуру MIPS) в бумажном виде, затем дополнение, которое применяет архитектуру ARM. Книга «Цифровая схемотехника и архитектура компьютера» стала настолько популярна, что ее начали использовать в ведущих российских вузах. Единственного, чего ей не хватало, это полноценного практического курса, который бы мог дополнить основной материал лабораторными работами. В 2019 г. такой курс был создан. Под эгидой МИЭМ НИУ ВШЭ была собрана большая команда преподавателей и разработчиков из СНГ и США, написавшие книгу «Цифровой синтез: практический курс» под редакцией А. Ю. Романова и Ю. В. Панчула. Книга хороша тем, что она раскрывает и дополняет материал книги Харрисов, а также поддержана репозитарием с исходными кодами всех примеров, приведенных в ней, и адаптирована под выполнение лабораторных работ на дешевых отладочных платах с ПЛИС.

И вот, наконец, ввиду все большего распространения архитектуры RISC-V, появилось новое издание книги «Цифровая схемотехника и архитектура компьютера».

Почему RISC-V?

Лицензируемые ядра RISC-процессоров совершили еще одну революцию в конце 1990 – начале 2000-х годов, когда ARM стал сердцем сотовых телефонов от Nokia и Ericsson, а MIPS стали использовать в телевизорах Sony, игровых приставках

и даже роботах. К компаниям ARM и MIPS присоединились несколько конкурентов, в частности ARC и Tensilica, которые образовали так называемую индустрию полупроводниковой интеллектуальной собственности, semiconductor IP, общим размеров в несколько миллиардов долларов.

Помимо разработчиков центральных процессоров в эту индустрию вошли Imagination Technologies – компания, которая спроектировала графический процессор PowerVR для ранних Apple iPhone, затем разработчик процессора для обработки сигналов CEVA и уже в наше время компании, которые выпускают ускорители нейросетевых вычислений.

ARM и MIPS получали доход двумя способами:

1) продажей лицензий на процессорные ядра – фактически на использование сотни тысяч строк на Verilog, написанных инженерами ARM и MIPS, внутри систем на кристалле заказчика. Примерами таких компаний стали Microchip, которая лицензировала ядро MIPS M4K для микроконтроллеров PIC32, и ST Microelectronics, которая лицензировала ядра ARM Cortex M для линейки микроконтроллеров STM32;
2) продажей так называемой архитектурной лицензии – права на создание процессора собственной микроархитектуры. Инженеры компании-покупателя архитектурной лицензии создавали собственную микроархитектуру и могли разрабатывать код на Verilog сами, но их ядро делалось совместимым по архитектуре (системе команд) с ARM или MIPS. Последним примером такого лицензиата является компания Apple, которая создала свое ARM-совместимое ядро для системы на кристалле Apple M1.

Хотя разделение компаний на разработчиков IP-блоков и разработчиков систем на кристалле помогло развить индустрию в 1990–2000-е годы, не все в этой схеме было идеальным.

- Во-первых, многие компании были недовольны условиями и политикой лицензирования как ядер, так и архитектуры. Особенно сильное негативное впечатление на индустрию произвел судебный процесс MIPS против Lexra в 1999 году, в результате которого пионер IP-лицензирования компания Lexra обанкротилась из-за довольно мелкого нарушения патента на редко используемые инструкции невыравненного обмена с памятью (https://www.eetimes.com/lexra-quits-ip-cores-business-in-deal-with-mips/).
- Во-вторых, контроль архитектуры со стороны коммерческих компаний не нравился университетским исследователям. Хотя MIPS активно использовался в учебниках, а ARM давал гранты университетам, но ученые были недовольны перспективой получения писем от корпоративных юристов из-за какого-нибудь созданного ими экспериментального процессора.
- Наконец, во всех RISC-архитектурах скопились разные черты, которые когда-то казались хорошими идеями, но стали тормозом прогресса при усложнении процессоров, повышении частоты, введении микроархитектуры с внеочередным выполнением команд, переменной длины инструкций и предсказателями перехода. У SPARC такой чертой были регистровые окна, у MIPS – слоты отло-

женного ветвления, у ARM – условное выполнение инструкций. Нужна была ревизия мира RISC-процессоров.

И этой ревизией стала RISC-V – архитектура, созданная в 2010 году группой того же Дэвида Паттерсона из Университета Калифорнии в Беркли, который написал два учебника и стоял у истоков архитектуры SPARC. Группа RISC-V не только объединила опыт процессорных компаний за предыдущие 30 лет, но и вступила в партнерство с Linux Foundation и многими крупными компаниями – Google, AMD, Western Digital.

Когда вы используете архитектуру RISC-V для проектирования своего процессора, вам не нужно платить за архитектурную лицензию. При этом сами вы можете получать за свой процессор деньги: продавать его как IP-блок, систему на кристалле или производить на его основе чипы. Вы также можете решить сделать бесплатный процессор с открытым кодом на Verilog для исследователей – это тоже поощряется сообществом вокруг архитектуры RISC-V.

Сейчас RISC-V может сыграть большую роль в становлении российской электроники. Российские компании CloudBEAR и Syntacore (приобретенная компанией «Ядро») работают над процессорами собственной микроархитектуры, совместимыми по системе команд с архитектурой RISC-V. Это идеальная комбинация, которая позволяет разрабатывать свои процессоры и конкурировать по производительности, энергопотреблению и набору расширений с производителями на мировом рынке, одновременно сохраняя программную совместимость со всеми программами, которые создаются для экосистемы RISC-V во всем мире. К таким программам относятся компиляторы, операционные системы и прикладные программы – от программ для миниатюрных чипов для интернета вещей до мобильных устройств, автомобильной электроники, десктопов и суперкомпьютеров.

Подводя итог

Предыдущие издания учебника Харрисов уже помогли исправить серьезный дисбаланс в преподавании цифровой электроники в России, который возник еще в 1990-е годы. Книга также стала отправной точкой для создания курса лабораторных работ на ПЛИС под эгидой МИЭМ НИУ ВШЭ, онлайн-курсов от РОСНАНО и семинаров на ChipEXPO в Сколково. Новое же издание учебника Харрисов выходит как раз тогда, когда в России разворачиваются амбициозные проекты по созданию высокопроизводительных процессорных ядер, которые совместимы с открытой международной архитектурой RISC-V и при этом спроектированы в России.

Мы ожидаем, что читатели этой книги станут топ-разработчиками и бизнес-лидерами российской электронной промышленности и помогут ей занять место в мире, которое соответствует российским традициям достижений в математике, физике, атомных и космических технологиях.

Юрий Панчул,
инженер-проектировщик CPU, GPU и сетевых микросхем,
с опытом работы в MIPS Technologies, Imagination Technologies,
Juniper Networks и Samsung Advanced Computing Lab

Предисловие от редактора русского перевода

Дорогие читатели,
перед вами – уникальное издание.

После распада СССР в русскоязычной образовательной среде возник вакуум, интеллектуальный рынок быстро захватили иностранные САПРы, а на первых ролях оказалась западная электроника. В сфере образования курсы по цифровой электронике нередко сводились к локальным курсам под конкретные платы, наличествовавшие в том или ином университете, а во многих случаях (в том числе из-за отсутствия надлежащего оборудования) учебный процесс превращался в сугубо теоретическое изучение дисциплины. Об этом явлении совершенно справедливо написал Юрий Панчул: https://habr.com/ru/post/589091/ («Почему книга Эндрю Таненбаума "Архитектура компьютера" вредна для образования»), чью точку зрения я полностью поддерживаю, поскольку сам учился по книге Таненбаума.

К счастью, в последующие годы картина начала меняться. Университеты стали богаче, появилась возможность приобретения необходимого оборудования, оно стало доступно и для личного пользования; началось оживление в российских компаниях, и обозначилась все большая потребность в специалистах по цифровой электронике. Все эти обстоятельства сформировали запрос на появление массовых учебных материалов на русском языке.

Звезды сошлись в 2016 году: для написания книги, по инициативе Юрия Панчула, удалось собрать вместе специалистов из ряда университетов и международных компаний, получить финансирование от Imagination technologies и найти понимание ведущего издательства в этой сфере – «ДМК Пресс». Основой для написания материала будущего издания стала великолепная книга D. M. Harris, S. L. Harris «Digital Design and Computer Architecture», де-факто являющаяся стандартом при изучении компьютерной архитектуры и цифрового синтеза во многих зарубежных университетах.

Так появилось первое издание книги «Цифровая схемотехника и архитектура компьютера». Несмотря на то что перевод был в некоторой степени аматорским и в первом издании обнаружились некоторые ошибки и неточности, книга стала бестселлером и разошлась тиражом в не одну тысячу экземпляров. Поскольку перевод первого издания книги осуществлялся без моего участия, мною в учебном процессе использовалась ее английская версия. Но как только появился русский перевод, он был сразу внедрен в учебные курсы, и на нем выросло несколько поколений студентов.

Дальше – больше. «Цифровой синтез» издали в цветном варианте, потом вышло дополнение по архитектуре ARM, а также была выпущена отдельная книга, допол-

няющая основную: «Цифровой синтез: практический курс», которая представляет собой компьютерный практикум, построенный на дешевых и доступных платах ПЛИС, при этом был сделан акцент на изучении языка Verilog.

Следует отметить, что время не стоит на месте: архитектура MIPS, которой посвящена исходная книга, все больше теряет свои позиции и вытесняется RISC-V, объединяющей в себе новые подходы к проектированию RISC-процессоров и принципы открытой разработки. Таким образом, появилась насущная необходимость в переводе нового издания книги D. M. Harris, S. L. Harris «Digital Design and Computer Architecture. RISC-V Edition». Хотя новое издание в целом ряде глав пересекается с исходной книгой, другие главы, посвященные архитектуре RISC-V, – полностью новые. Можно было пойти при этом по одному из путей: либо выпустить дополнение к основной книге (как это было сделано для архитектуры ARM), либо перевыпустить книгу полностью. Чтобы не нарушать целостность произведения, было принято решение пойти по второму пути, попутно исправив допущенные ранее ошибки и тщательно переработав старые главы. Результат этого труда – перед вами.

Данная книга будет полезна всем студентам (таких вузов, как, например, МИЭТ или ИТМО), изучающим архитектуру компьютера и языки описания аппаратуры, а также всем разработчикам, которым необходимо понимать, как устроен микропроцессор / микроконтроллер или другая цифровая схема изнутри.

Александр Юрьевич Романов,
научный редактор книги,
к. т. н., доцент МИЭМ НИУ ВШЭ,
преподаватель курсов «Проектирование систем на кристалле»
и «Системное проектирование цифровых устройств»,
г. Москва, Россия

Предисловие

Эта книга уникальна тем, что описывает цифровую схемотехнику с точки зрения компьютерной архитектуры, начиная с двоичной логики и заканчивая проектированием микропроцессора.

Мы считаем, что проектирование микропроцессора является своеобразным обрядом посвящения для студентов инженерных и компьютерных специальностей. Внутренняя работа микропроцессора кажется почти магической для непосвященных, но при подробном объяснении оказывается простой и доступной для понимания. Проектирование цифровых схем само по себе является захватывающим предметом. Программирование на языке ассемблера позволяет понять внутренний язык, на котором говорит микропроцессор. Микроархитектура, в свою очередь, является тем связующим звеном, которое объединяет эти предметы воедино.

Первые две версии этого набирающего популярность учебника описывают архитектуры MIPS и ARM. MIPS – одна из исходных вычислительных архитектур с сокращенным набором команд (Reduced Instruction Set Computing, RISC), простая в изучении и применении. Значимость архитектуры MIPS сложно переоценить, поскольку она вдохновила разработчиков на создание последующих архитектур, включая RISC-V. Архитектура ARM стала очень популярной за последние несколько десятилетий благодаря своей эффективности и богатой экосистеме. Было продано более 50 млрд процессоров ARM, и более 75 % людей на планете используют продукты с этими процессорами.

В течение последнего десятилетия архитектура RISC-V становится все более значимой как с образовательной, так и с коммерческой точки зрения. Будучи широко распространенной компьютерной архитектурой с открытым исходным кодом, RISC-V сочетает простоту MIPS с гибкостью и функциональностью современных процессоров.

С познавательной точки зрения использование трех версий учебника – MIPS, ARM и RISC-V – полностью идентично. Архитектура RISC-V имеет ряд особенностей, включающих расширяемость и компактный формат представления инструкций, которые повышают ее эффективность, но немного увеличивают сложность. Три микроархитектуры также похожи, а архитектуры MIPS и RISC-V имеют много общего. Мы планируем переиздавать версии учебника про MIPS, ARM и RISC-V до тех пор, пока эти архитектуры востребованы рынком.

Особенности книги

Эта книга содержит ряд особенностей.

Одновременное использование языков SystemVerilog и VHDL

Языки описания аппаратуры (hardware description languages, HDL) находятся в основе современных методов проектирования сложных цифровых систем. К сожалению, разработчики делятся на две примерно равные группы, использующие два

разных языка, – SystemVerilog и VHDL. Языки описания аппаратуры рассматриваются в **главе 4**, сразу после глав, посвященных проектированию комбинационных и последовательностных логических схем. Затем языки HDL используются в **главах 5** и **7** для разработки цифровых блоков большего размера и процессора целиком. Тем не менее **главу 4** можно безболезненно пропустить, если изучение языков HDL не входит в программу.

Эта книга уникальна тем, что использует одновременно и SystemVerilog, и VHDL, что позволяет читателю освоить проектирование цифровых систем сразу на двух языках. В **главе 4** сначала описываются общие принципы, применимые к обоим языкам, а затем вводится синтаксис и приводятся примеры использования этих языков. Этот двуязычный подход облегчает преподавателю выбор языка HDL, а читателю позволит перейти с одного языка на другой как во время учебы, так и в профессиональной деятельности.

Архитектура и микроархитектура процессора RISC-V

Главы 6 и **7** посвящены изучению архитектуры и микроархитектуры RISC-V. Архитектура RISC-V является идеальным учебным пособием в том смысле, что это реальная архитектура, на которой основаны миллионы выпускаемых ежегодно микросхем, и в то же время она проста для изучения. Кроме того, сотни университетов по всему миру разрабатывают учебные курсы, лабораторные работы и различные инструменты именно для этой архитектуры.

Живые примеры

В дополнение к обсуждению основной темы этого учебника – архитектуры RISC-V – в **разделе 6.8** для расширения кругозора студентов рассматривается архитектура процессоров Intel x86. В **главе 9** (доступной в виде онлайн-приложения) также описываются периферийные устройства на примере популярной платы для разработки RED-V RedBoard от SparkFun, в основе которой лежит процессор SiFive Freedom E310 RISC-V. Эти живые примеры показывают, как описанные в данных главах концепции применяются в реальных микросхемах, которые широко используются в персональных компьютерах и бытовой электронике.

Доступное описание высокопроизводительных архитектур

Глава 7 содержит краткий обзор особенностей современных высокопроизводительных микроархитектур, включая такие, как внеочередное выполнение команд, суперскалярность, многопоточность и многоядерность. Материал изложен в доступной для первокурсников форме и показывает, как можно расширить микроархитектуры, описанные в книге, чтобы получить современный процессор.

Упражнения в конце глав и вопросы для собеседования

Лучшим способом изучения цифровой схемотехники является разработка устройств. В конце каждой главы приведены многочисленные упражнения. За

упражнениями следует набор вопросов для собеседования, которые наши коллеги обычно задают студентам, претендующим на работу в отрасли. Эти вопросы предлагают читателю взглянуть на задачи, с которыми соискателям придется столкнуться в ходе собеседования при трудоустройстве. Решения упражнений доступны через веб-сайт книги и специальный веб-сайт для преподавателей. Более подробная информация приведена в следующем разделе.

Материалы в сети Интернет

Дополнительные англоязычные материалы для этой книги доступны на веб-сайте по адресу http://www.ddcabook.com или на сайте издателя: https://www.elsevier.com/books-and-journals/book-companion/9780128200643. Эти веб-сайты доступны для всех читателей и содержат следующие материалы:

- ссылки на видеокурсы;
- решения упражнений с нечетными номерами;
- иллюстрации из книги в форматах PDF и PPTX;
- ссылки на профессиональные инструменты автоматизированного проектирования (САПР) от Intel®;
- инструкции по использованию PlatformIO (расширение Visual Studio Code) для компиляции, сборки и моделирования кода на языках C и ассемблера для процессоров RISC-V;
- HDL-код для процессора RISC-V;
- полезные советы по использованию Intel Quartus;
- слайды лекций в формате PowerPoint (PPTX);
- образцы учебных и лабораторных материалов для курса;
- список опечаток и исправлений.

Также существует специальный веб-сайт для преподавателей, зарегистрировавшихся на https://inspectioncopy.elsevier.com, который содержит:

- решения всех упражнений;
- решения заданий к лабораторным работам.

Открытые курсы на EdX

К этой книге прилагаются открытые курсы на сайте EdX (https://www.edx.org/). Курсы содержат видеолекции, интерактивные упражнения, а также интерактивные наборы задач и лабораторные работы. Набор курсов состоит из двух частей – «Цифровая схемотехника» (ENGR 85A) и «Компьютерная архитектура» (ENGR 85B), – разработанных в Harvey Mudd College (HarveyMuddX; на EdX выполните поиск по фразам «Digital Design HarveyMuddX» и «Computer Architecture HarveyMuddX»). Вам не придется платить за просмотр видео, но EdX взимает плату за интерактивные упражнения и сертификат. Для студентов предусмотрены скидки.

Как использовать программный инструментарий в учебном курсе

Программное обеспечение Intel Quartus

Программное обеспечение Quartus Web Edition и Lite Edition представляет собой бесплатные версии профессиональной САПР Intel Quartus™, предназначенной для разработки устройств на FPGA. Это позволяет студентам проектировать цифровые устройства в виде принципиальных схем или на языках SystemVerilog и VHDL. После создания схемы или кода устройства студенты могут моделировать их поведение с использованием САПР ModelSim™-Intel FPGA Edition или Starter Edition, которые входят в состав САПР Intel Quartus. Quartus также содержит встроенный инструмент логического синтеза, который поддерживает языки описаний SystemVerilog и VHDL.

Разница между Web Edition, Lite Edition и Pro Edition заключается в том, что Web- и Lite Edition поддерживают только подмножество наиболее распространенных FPGA производимых Intel FPGA (Altera). Бесплатные версии ModelSim искусственно снижают производительность моделирования для проектов, содержащих больше 10 тысяч строк HDL-кода, тогда как профессиональная версия ModelSim этого не делает.

PlatformIO

Расширение PlatformIO для редактора Visual Studio Code представляет собой набор средств разработки программного обеспечения (software development kit, SDK) для RISC-V. Поскольку появление каждой новой платформы влекло за собой появление нового SDK, расширение PlatformIO сделало процесс программирования и использования различных процессоров заметно проще благодаря наличию унифицированного интерфейса для большого количества платформ и устройств. SDK PlatformIO можно скачать бесплатно и использовать с RED-V RedBoard SparkFun, как описано в лабораторных работах на веб-сайте[1]. PlatformIO предоставляет доступ к коммерческому компилятору RISC-V и позволяет студентам разрабатывать программы на языках C и ассемблера, компилировать их, а затем запускать и выполнять их отладку на RedBoard SparkFun RED-V (глава 9 и соответствующие лабораторные работы).

Симулятор ассемблера Venus

Симулятор Venus, доступный по адресу https://www.kvakil.me/venus/, – это веб-симулятор ассемблера RISC-V. Программы разрабатываются (или копируются/вставляются) на вкладке **Редактор**, а затем моделируются и запускаются на вкладке **Симулятор**. Во время работы программы можно просматривать содержимое регистров и памяти.

[1] https://docs.platformio.org/en/latest/boards/sifive/sparkfun_redboard_v.html.

Лабораторные работы

Веб-сайт книги содержит ссылки на ряд лабораторных работ, которые охватывают все темы, начиная от проектирования цифровых систем и заканчивая архитектурой компьютера. Из лабораторных работ студенты узнают, как использовать САПР Quartus для описания своих проектов, их моделирования, синтеза и реализации. Лабораторные работы также включают темы по программированию на языке C и языке ассемблера с использованием PlatformIO и RedBoard RED-V от SparkFun.

После синтеза схемы студенты могут реализовать свои проекты, используя платы Altera DE2, DE2-115, DE0 или другую плату FPGA. Лабораторные работы подготовлены для плат DE2 или DE-115. Эти мощные и относительно недорогие платы доступны для заказа на сайте www.de2-115.terasic.com. На платах размещаются микросхемы FPGA, которые можно сконфигурировать для реализации студенческих проектов. Мы предоставляем лабораторные работы, которые описывают, как реализовать различные блоки на плате DE2-115 с помощью программного обеспечения Quartus.

Для проведения лабораторных работ учащимся необходимо загрузить и установить Intel Quartus Web или Lite Edition и Visual Studio Code с расширением PlatformIO. Преподаватели также могут установить эти САПР в учебных лабораториях. Лабораторные работы включают инструкции по разработке проектов на плате DE2/DE2-115. Этап практической реализации проекта на плате можно пропустить, но мы считаем, что он имеет большое значение для получения практических навыков.

Мы протестировали лабораторные работы на ОС Windows, но такие же инструменты доступны и для ОС Linux.

Курсы RVfpga

После изучения материала данной книги мы рекомендуем пройти бесплатный цикл из двух курсов RISC-V FPGA (RVfpga). Первый курс рассказывает о том, как сконфигурировать коммерческое ядро RISC-V для реализации на FPGA, запрограммировать его с помощью языка ассемблера RISC-V или C, добавить к нему периферийные устройства, а также проанализировать и изменить ядро и систему памяти, включая добавление инструкций в ядро. В этом курсе используется система на кристалле (SoC) SweRVolf с открытым исходным кодом (https://github.com/chipsalliance/Cores-SweRVolf), основанная на коммерческом ядре SweRV EH1 от Western Digital (https://www.westerndigital.com/solutions/business/risc-v). В курсе также показано, как использовать симулятор HDL с открытым исходным кодом Verilator и симулятор набора команд RISC-V с открытым исходным кодом Whisper от Western Digital. Второй курс, RVfpga-SoC, показывает, как построить SoC на основе SweRVolf, используя такие функциональные элементы, как ядро SweRV EH1, межмодульные соединения и память. Затем курс рассказывает пользователю о загрузке и запуске операционной системы Zephyr на SoC RISC-V. Все необходимое программное обеспечение и исходный код системы (файлы Verilog/SystemVerilog) бесплатны, а кур-

сы можно проходить с использованием симулятора, поэтому вам не придется покупать оборудование. Материалы RVfpga свободно доступны после регистрации на сайте программы Imagination Technologies University по адресу https://university.imgtec.com/rvfpga/.

Опечатки

Все опытные программисты знают, что любая сложная программа непременно содержит ошибки. Так же происходит и с книгами. Мы старались выявить и исправить все ошибки и опечатки в этой книге. Тем не менее некоторые ошибки могли остаться. Список найденных ошибок будет опубликован на веб-сайте книги.

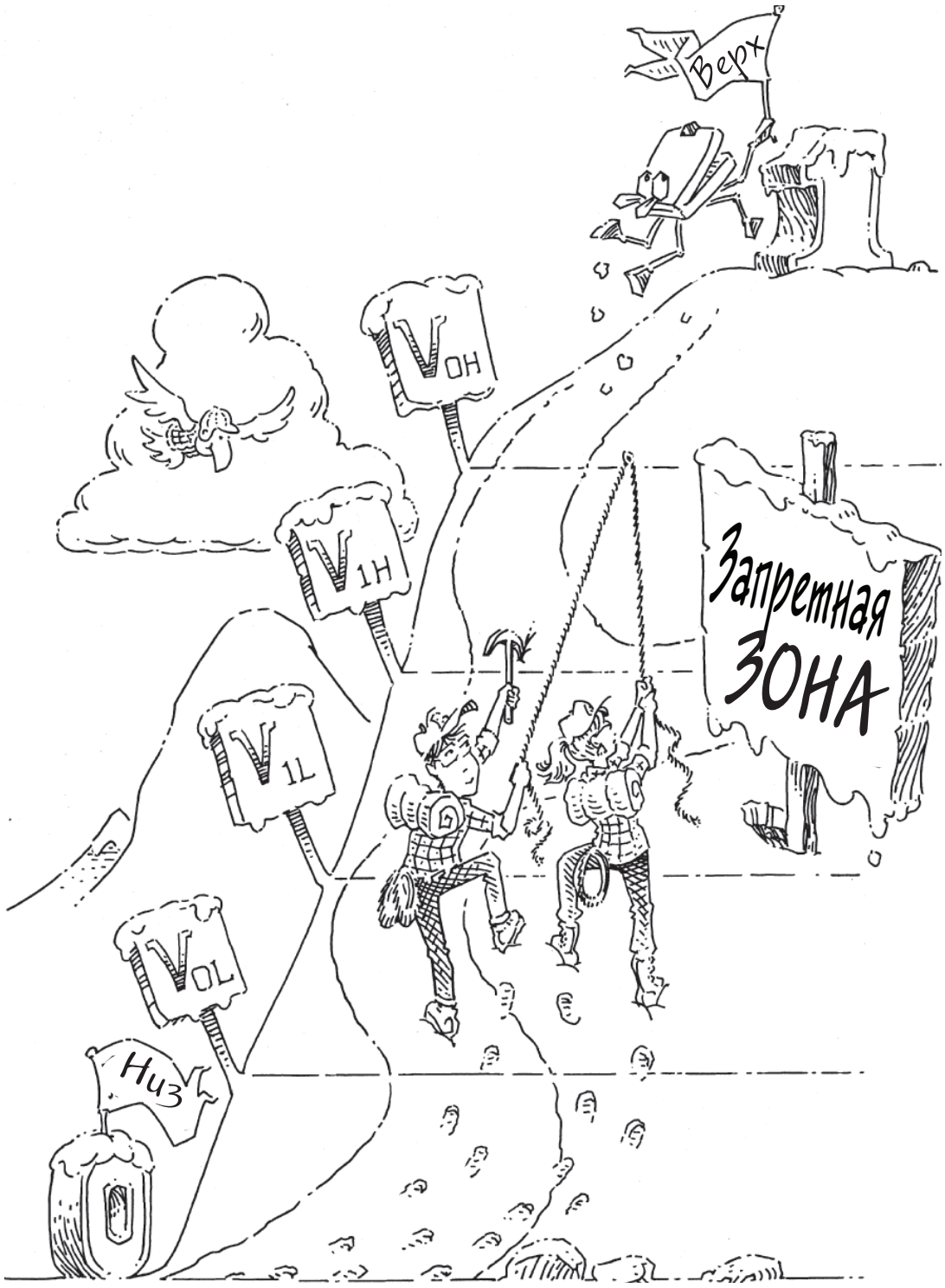
Пожалуйста, присылайте найденные ошибки по адресу ddcabugs@gmail.com (для английской версии книги; для русской версии – присылайте научному редактору русского перевода А. Ю. Романову на электронную почту a.romanov@hse.ru). Первый человек, который сообщит об ошибке в английском издании и предоставит исправление, которое мы используем в будущем переиздании книги, будет вознагражден премией в 1 доллар!

Признательность за поддержку

Мы высоко ценим огромный вклад Стива Меркена (Steve Merken), Нейта Макфаддена (Nate McFadden), Руби Гаммелл (Ruby Gammell), Андрэ Аке (Andrae Akeh), Маникандана Чандрасекарана (Manikandan Chandrasekaran) и остальных членов издательской команды Morgan Kaufmann, которые сделали эту книгу реальностью. Мы любим творчество Дуэйна Бибби (Duane Bibby), чьи забавные рисунки украшают страницы книги.

Мы хотели бы поблагодарить Мэтью Уоткинса (Matthew Watkins), который помог написать раздел о гетерогенных многопроцессорных системах в **главе 7**, и Джоша Брейка (Josh Brake), принявшего участие в написании **главы 9** о встроенных системах ввода-вывода. Мы высоко ценим работу Матео Марковича (Mateo Markovic) и Джорди Райдера (Geordie Ryder), которые рецензировали книгу и внесли свой вклад в решения упражнений. Огромный вклад в улучшение качества книги внесли многочисленные рецензенты: Дэниел Чавер Мартинес (Daniel Chaver Martinez), Рой Кравиц (Roy Kravitz), Звонимир Бандич (Zvonimir Bandic), Джузеппе Ди Луна (Giuseppe Di Luna), Штеффен Пол (Steffen Paul), Рави Миттал (Ravi Mittal), Дженнифер Виникус (Jennifer Winikus), Хешам Омран (Hesham Omran), Анхель Солис (Angel Solis), Райнер Дизон (Reiner Dizon) и Олоф Киндгрен (Olof Kindgren). Мы также очень признательны нашим студентам в колледже Harvey Mudd и Университете Невады в Лас-Вегасе, которые дали нам полезные отзывы на черновики этого учебника.

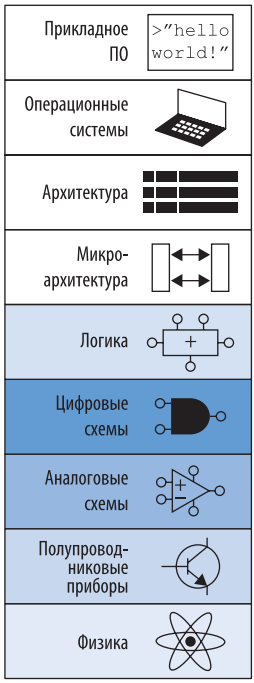
И конечно же, мы оба благодарим наши семьи за их любовь и поддержку.


Верх
V_{OH}
V_{1H}
V_{1L}
V_{OL}
Низ
Запретная
ЗОНА

Глава 1

От нуля до единицы

1.1. План игры
1.2. Искусство управления сложностью
1.3. Цифровая абстракция
1.4. Системы счисления
1.5. Логические элементы
1.6. За пределами цифровой абстракции
1.7. КМОП-транзисторы
1.8. Потребляемая мощность
1.9. Краткий обзор главы 1 и того, что нас ждет впереди
Упражнения
Вопросы для собеседования



1.1. План игры

За последние тридцать лет микропроцессоры буквально изменили наш мир до неузнаваемости. Ноутбук сейчас обладает большей вычислительной мощностью, чем большой компьютер из недавнего прошлого, занимавший целую комнату. Внутри современного автомобиля представительского класса можно обнаружить около пятидесяти микропроцессоров. Именно прогресс в области микропроцессорной техники сделал возможным появление сотовых телефонов и сети Интернет, значительно продвинул вперед медицину и радикально изменил тактику и стратегию современных войн. Объем продаж мировой полупроводниковой промышленности вырос с 21 млрд долларов в 1985 году до 300 млрд долларов в 2011 году, причем микропроцессоры составили львиную долю этих продаж. Мы убеждены, что микропроцессоры важны не только с технической, экономической и социальной точек зрения, но и стали одним из самых увлекательных изобретений в истории человечества. Когда вы за-

кончите чтение этой книги, вы будете знать, как разработать и построить ваш собственный микропроцессор, а навыки, полученные на этом пути, пригодятся вам для разработки многих других цифровых систем.

Мы предполагаем, что у вас уже есть базовые знания по теории электричества, некоторый опыт программирования и искреннее желание понять, что происходит внутри компьютера. В этой книге основное внимание уделяется разработке цифровых систем, то есть систем, которые используют для своей работы два уровня напряжения, представляющих единицу и ноль. Мы начнем с простейших цифровых логических элементов (digital logic gates), которые принимают определенную комбинацию единиц и нулей на входах и трансформируют ее в другую комбинацию единиц и нулей на выходах. После этого мы с вами научимся объединять эти простейшие логические элементы в более сложные модули, такие как сумматоры и блоки памяти. Затем перейдем к программированию на языке ассемблера – родном языке микропроцессора. И в завершение из кирпичиков логических элементов мы соберем полноценный микропроцессор, способный выполнять программы, разработанные на языке ассемблера.

Огромным преимуществом цифровых систем над аналоговыми является то, что необходимые для их построения блоки чрезвычайно просты, поскольку оперируют не непрерывными сигналами, а единицами и нулями.

Построение цифровой системы не требует запутанных математических расчетов или глубоких знаний в области физики. Вместо этого задача, стоящая перед разработчиком цифровых устройств, заключается в том, чтобы собрать сложную работающую систему из этих простых блоков. Возможно, микропроцессор станет первой разработанной вами системой, настолько сложной, что ее невозможно целиком удержать в голове. Именно поэтому одной из тем, проходящих красной нитью через эту книгу, является искусство управления сложностью системы.

1.2. Искусство управления сложностью

Одной из характеристик, отличающих профессионального инженера-электронщика или программиста от дилетанта, является систематический подход к управлению сложностью многоуровневой системы. Современные цифровые системы построены из миллионов и миллиардов транзисторов. Человеческий мозг не в состоянии предсказать поведение подобных систем путем составления уравнений, описывающих движение каждого электрона в каждом транзисторе системы, и последующего решения этой системы уравнений. Для того чтобы разработать удачный

микропроцессор и не утонуть при этом в море избыточной информации, необходимо научиться управлять сложностью разрабатываемой системы.

1.2.1. Абстракция

Критически важный принцип управления сложностью системы – *абстракция*, подразумевающая исключение из рассмотрения тех элементов, которые в данном конкретном случае несущественны для понимания работы этой системы. Любую систему можно рассматривать с различных уровней абстракции. Политику, участвующему в выборах, например, нет нужды учитывать все детали окружающего его мира, ему достаточно абстрактной иерархической модели страны, состоящей из населенных пунктов, областей и федеральных округов. В области может быть несколько населенных пунктов, а федеральный округ включает в себя разные области. Если политик борется за пост президента, то его, скорее всего, интересует то, как проголосует федеральный округ в целом, при этом ему не обязательно знать, какое количество голосов он наберет в каждом конкретном населенном пункте этого округа. Для политика федеральный округ – это его уровень абстракции. С другой стороны, бюро переписи населения обязано знать количество жителей в каждом городе или поселке страны и потому должно оперировать на самом низком уровне абстракции данной системы – на уровне населенных пунктов.

На **рис. 1.1** показаны уровни абстракции, типичные для любой электронной компьютерной системы вместе со строительными блоками, характерными для каждого уровня абстракции этой системы. На самом низком уровне абстракции находится физика, изучающая движение электронов. Поведение электронов описывается квантовой механикой и системой уравнений Максвелла.

Рассматриваемая нами современная электронная система состоит из полупроводниковых устройств (devices), таких как транзисторы (а когда-то это были электронные лампы). Каждое такое устройство имеет четко определенные точки соединения с другими подобными устройствами. Эти точки мы будем называть *контактами* (в англоязычной литературе используется термин *terminal*). Любое электронное устройство может быть представлено абстрактной математической моделью, описывающей изменяющуюся во времени взаимозависимость тока и напряжения. Такие же изменения тока и напряжения можно наблюдать на экране осциллографа, если подключить осциллограф к контактам реального устройства. Данный под-

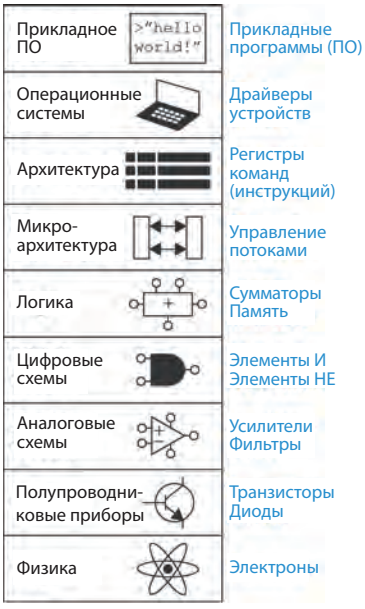


Рис. 1.1 Уровни абстракции электронной вычислительной системы

Каждая глава этой книги начинается с иконок (рис. 1.1), символически изображающих уровни абстракции электронной системы, которые мы перечислили выше. Иконка темно-синего цвета указывает на тот уровень абстракции, которому уделяется главное внимание в этой конкретной главе. Иконки более светлого оттенка синего указывают на другие уровни абстракции, также затронутые в главе.

ход означает, что если рассматривать систему на уровне устройств, функции которых однозначно определены, то можно не учитывать поведение электронов внутри отдельных устройств этой системы.

Следующий уровень абстракции – это *аналоговые схемы* (analog circuits), в которых полупроводниковые устройства соединены таким образом, чтобы они образовывали функциональные компоненты, например усилители. Напряжение на входе и на выходе аналоговой цепи изменяется в непрерывном диапазоне.

В отличие от аналоговых цепей, *цифровые схемы* (digital circuits), такие как логические элементы, используют два строго ограниченных дискретных уровня напряжения. Один из этих дискретных уровней – это логический ноль, другой – логическая единица. В разделах этой книги, посвященных разработке цифровых схем и устройств, мы будем использовать простейшие цифровые схемы для построения сложных цифровых модулей, таких как сумматоры и блоки памяти.

Микроархитектурный уровень абстракции, или просто *микроархитектура* (microarchitecture), связывает логический и архитектурный уровни абстракции. Архитектурный уровень абстракции, или *архитектура* (architecture), описывает компьютер с точки зрения программиста. Например, архитектура Intel x86, используемая микропроцессорами большинства персональных компьютеров (ПК), определяется набором инструкций и регистров (памяти для временного хранения переменных), доступным для использования программистом. Микроархитектура – это соединение простейших цифровых элементов в логические блоки, предназначенные для выполнения команд, определенных какой-то конкретной архитектурой. Отдельно взятая архитектура может быть реализована с использованием различных вариантов микроархитектур с разным соотношением цены, производительности и потребляемой энергии, и такое соотношение зачастую выбирается как баланс между этими тремя факторами. Процессоры Intel Core i7, Intel 80486 и AMD Athlon, например, используют одну и ту же архитектуру x86, но реализованную с применением трех разных микроархитектурных решений.

Рассмотрим область программного обеспечения. *Операционная система* (operating system) управляет операциями нижнего уровня, такими как доступ к жесткому диску или управление памятью. И наконец, программное обеспечение использует ресурсы операционной системы для решения конкретных задач пользователя.

Именно принцип *абстрагирования от маловажных деталей* позволяет вашей бабушке общаться с внуками в интернете, не задумываясь о квантовых колебаниях электронов или организации памяти компьютера.

Предмет этой книги – уровни абстракции от цифровых схем до компьютерной архитектуры. Работая на каком-либо из этих уровней абстракции, полезно знать кое-что и об уровнях абстракции, непосредственно сопряженных с тем уровнем, где вы находитесь. Программист, например, не сможет полностью оптимизировать код без понимания архитектуры процессора, который будет выполнять эту программу. Инженер-электронщик, разрабатывающий какой-либо блок микросхемы, не сможет найти компромисс между быстродействием и уровнем потребления энергии транзисторами, ничего не зная о той цифровой схеме, где этот блок будет использоваться. Мы надеемся, что к тому времени, когда вы закончите чтение этой книги, вы сможете выбрать уровень абстракции, необходимый для успешного выполнения любой стоящей перед вами задачи, и оценить влияние ваших инженерных решений на другие уровни абстракции в разрабатываемой вами системе.

1.2.2. Конструкторская дисциплина

Конструкторская дисциплина – это преднамеренное ограничение самим конструктором выбора возможных вариантов разработки, что позволяет работать продуктивнее на более высоком уровне абстракции. Использование взаимозаменяемых частей – это, вероятно, самый хорошо знакомый всем нам пример практического применения конструкторской дисциплины. Одним из первых примеров использования взаимозаменяемых деталей и узлов стала унификация при производстве кремневых ружей. До начала XIX века такие ружья производились вручную и в штучном порядке. Высококвалифицированный оружейный мастер тщательно подтачивал и подгонял комплектующие, произведенные несколькими не связанными друг с другом ремесленниками. Конструкторская дисциплина для обеспечения взаимозаменяемости деталей и узлов произвела революцию в оружейной промышленности. Ограничение ассортимента комплектующих деталей до стандартного набора с жестко установленными допусками для каждой детали позволило собирать и ремонтировать ружья гораздо быстрее и использовать при этом менее квалифицированный персонал. Оружейный мастер перестал тратить свое время на разрешение проблем, связанных с нижними уровнями абстракции, такими как доводка какого-то конкретного ствола или исправление формы отдельного взятого приклада.

В контексте данной книги соблюдение конструкторской дисциплины в виде максимального использования цифровых схем играет очень важную роль. В цифровых схемах используются дискретные значения напряжения, в то время как в аналоговых схемах напряжение изменяется непрерывно. Таким образом, цифровые схемы, которые можно рассматривать как подмножество аналоговых цепей, в некотором смысле уступают по своим характеристикам более широкому классу аналого-

вых цепей. Но цифровые цепи гораздо проще разработать. Ограничивая использование аналоговых схем и по возможности заменяя их цифровыми, мы можем легко объединять отдельные компоненты в сложные системы, которые в конечном итоге для большинства приложений превзойдут по своим параметрам системы, построенные на аналоговых цепях. Примером тому могут служить цифровые телевизоры, компакт-диски (CD) и мобильные телефоны, которые уже практически полностью вытеснили своих аналоговых предшественников.

Капитан Мериуэзер Льюис — один из руководителей знаменитой экспедиции Льюиса и Кларка на северо-запад США, был, пожалуй, одним из самых ранних сторонников взаимозаменяемости. В 1806 году в своем дневнике, описывая особенности кремниевых унификации деталей кремневых ружей того времени, он написал следующее:

> «Ружья Дрюера и сержанта Прайора одновременно вышли из строя. На ружье Дрюера сломался ударно-спусковой механизм, и мы заменили его на новый. У ружья сержанта Прайора был сломан курковый винт, вместо которого мы поставили запасной, заранее изготовленный специально для ударно-спускового механизма этого ружья на мануфактуре Харперс Фейри, где это оружие и было произведено. Если бы не предусмотрительность, заключавшаяся в том, что мы заранее позаботились о запасных частях для ружей, и не мастерство Джона Шилдса, выполнившего всю работу, то большинство ружей нашей экспедиции к этому времени было бы полностью непригодно для какого-либо использования. И я имею полное право записать в своем дневнике, что, к счастью для нас, все наше оружие находится в прекрасном состоянии».

См.: История экспедиции Льюиса и Кларка: в 4 т. / под ред. Элиота Куэса. 1-е изд.: Харпер, Нью-Йорк, 1893; Переизд.: Довер, Нью-Йорк (3 тома), 3:817.

1.2.3. Три базовых принципа

В дополнение к абстрагированию от несущественных деталей и конструкторской дисциплине разработчики электронных систем используют еще три базовых принципа для управления сложностью системы: иерархичность, модульность конструкции и регулярность. Эти принципы применимы как к программному обеспечению, так и к аппаратной части компьютерных систем.

- *Иерархичность* – принцип иерархичности предполагает разделение системы на отдельные модули, а затем последующее разделение каждого такого модуля на фрагменты до уровня, позволяющего легко понять поведение каждого конкретного фрагмента.
- *Модульность* – принцип модульности требует, чтобы каждый модуль в системе имел четко определенную функциональность и набор интерфейсов и мог быть легко и без непредвиденных побочных эффектов соединен с другими модулями системы.
- *Регулярность* – принцип регулярности требует соблюдения единообразия при разработке отдельных модулей системы. Стандартные модули общего назначения, например такие как блоки питания, могут использоваться многократно, во много раз снижая количество модулей, необходимых для разработки новой системы.

Для иллюстрации трех базовых принципов вновь воспользуемся аналогией из оружейного производства. Нарезное кремневое ружье было одним из самых сложных устройств массового применения в начале XIX века. Используя принцип иерархичности, мы можем разделить его на три главных модуля, как показано на **рис. 1.2**: ствол, ударно-спусковой механизм и приклад с цевьем.

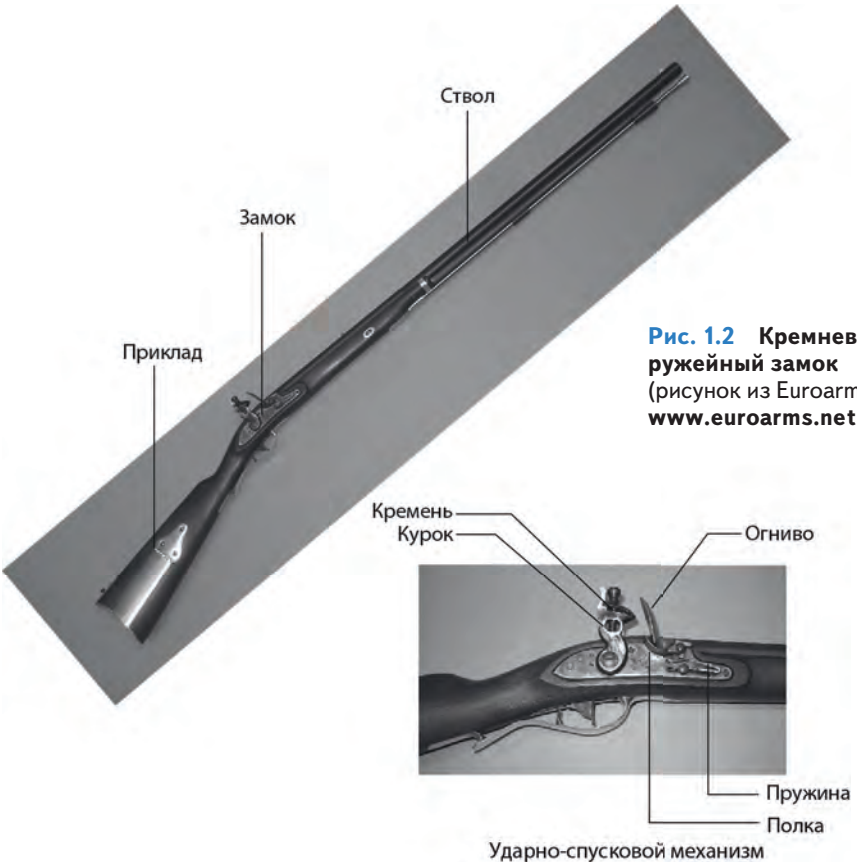


Рис. 1.2 Кремневый ружейный замок (рисунок из Euroarms Italia **www.euroarms.net** © 2006 г.)

Ствол – это длинная металлическая труба, через которую при выстреле выбрасывается пуля. Ударно-спусковой механизм производит выстрел. Деревянные приклад и цевье соединяют воедино остальные части ружья и обеспечивают стрелку надежное удержание оружия при выстреле. В свою очередь, ударно-спусковой механизм включает в себя спусковой крючок, курок, кремень, огниво и пороховую полку. Каждый из этих компонентов также может рассматриваться как следующий иерархический уровень и может быть разделен на более мелкие детали.

Принцип модульности требует, чтобы каждый компонент выполнял четко определенную функцию и имел интерфейс. Функция приклада и цевья – служить базой для установки ствола и ударно-спускового механизма. Интерфейс для приклада и цевья – это их длина и расположение крепежных элементов, таких как винты или шурупы. Ствол ружья, изготовленного с соблюдением принципа модульности конструкции, может быть установлен на приклады и цевья от разных производителей, если все соединяемые части имеют правильную длину и подходящие

крепежные элементы. Функция ствола – разогнать пулю до необходимой скорости и придать ей вращение, чтобы увеличить точность стрельбы[1]. Принцип модульности требует также, чтобы при соединении модулей не возникало никаких побочных эффектов: конструкция приклада и цевья не должна препятствовать функционированию ствола.

Принцип регулярности учит тому, что взаимозаменяемые детали – это хорошая идея. При соблюдении принципа регулярности поврежденный ствол может быть с легкостью заменен на аналогичный. Стволы могут изготовляться на поточной линии с гораздо большей экономической эффективностью, чем в случае штучного производства.

В данной книге мы будем постоянно возвращаться к этим трем базовым принципам: иерархичности, модульности и регулярности.

1.3. Цифровая абстракция

Чарльз Бэббидж
1791–1871

Чарльз Бэббидж родился в 1791 году. Закончил Кембриджский университет и женился на Джорджиане Витмур. Он изобрел аналитическую машину – первый в мире механический компьютер. Чарльз Бэббидж также изобрел предохранительную решетку для локомотивов, спидометр и универсальный почтовый тариф. Ученый также очень интересовался отмычками для замков и почему-то ненавидел уличных музыкантов. (Портрет любезно предоставлен Fourmilab Швейцария, www.fourmilab.ch.)

Большинство физических величин изменяются непрерывно. Например, напряжение в электрическом проводе, частота колебаний или распределение массы – все это параметры, изменяющиеся непрерывно. Цифровые системы, с другой стороны, представляют информацию в виде дискретно меняющихся переменных с конечным числом строго определенных значений.

Одной из наиболее ранних цифровых систем стала аналитическая машина Чарльза Бэббиджа, которая использовала переменные с десятью дискретными значениями. Начиная с 1834 года и до 1871 года[2] Бэббидж разрабатывал и пытался построить этот механический компьютер. Шестеренки аналитической машины могли находиться в одном из десяти фиксированных положений, а каждое такое положение было промаркировано от 0 до 9, подобно механическому счетчику пробега автомобиля. **Рисунок 1.3** показывает, как выглядел прототип аналитической машины. Каждый ряд шестеренок такой машины обрабатывал одну цифру. В своем механическом компьютере Бэббидж использовал 25 рядов шестеренок таким образом, чтобы машина обеспечивала вычисления с точностью до 25-го знака.

В отличие от машины Бэббиджа большинство электронных компьютеров использует двоичный (бинарный) код. В случае двоичного кода высокое напряжение – это единица, а низкое напряжение – ноль, поскольку гораздо легче оперировать двумя уровнями напряжения, чем десятью.

---

[1] Кремневые ружья не были нарезными и использовали круглые пули. – *Прим. перев.*

[2] А большинству из нас кажется, что обучение в университете – это так долго!

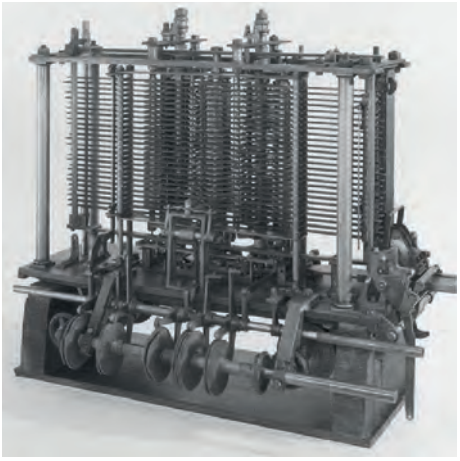


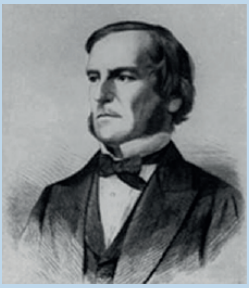
Рис. 1.3 Аналитическая машина Бэббиджа в год его смерти (1871)
(изображение любезно предоставлено Музеем науки и общества)

Объем информации D, передаваемый одной дискретной переменной, которая может находиться в N различных состояниях, измеряется в единицах, называемых *битами,* и вычисляется по следующей формуле:

$$D = \log_2 N \text{ бит.} \tag{1.1}$$

Двоичная переменная передает $\log_2 2 = 1$ – один бит информации. Теперь вам, вероятно, понятно, почему единица информации называется битом. Бит (bit) – это сокращение от английского *binary digit*, что дословно переводится как *двоичный разряд*. Каждая шестеренка в машине Бэббиджа содержит $\log_2 10 = 3{,}322$ бита информации, поскольку она может находиться в одном из $2^{3{,}322} = 10$ уникальных положений. Теоретически непрерывный сигнал может передавать бесконечное количество информации, поскольку может принимать неограниченное число значений. На практике шум и ошибки измерения ограничивают информацию, передаваемую большинством непрерывных сигналов, диапазоном от 10 бит до 16 бит. Если же измерение уровня сигнала должно быть произведено очень быстро, то объем передаваемой информации будет еще ниже (в случае 10 бит, например, это будет только 8 бит).

Предмет этой книги – цифровые схемы, использующие двоичные переменные ноль и единицу. Джордж Буль разработал систему логики, использующую двоичные переменные, и эту систему сегодня называют его именем – *булева логика*. Логические переменные могут принимать значения *ИСТИНА* (TRUE) или *ЛОЖЬ* (FALSE). В электронных компьютерах положительное напряжение обычно представляет единицу, а нулевое напряжение представляет ноль. В этой книге мы будем использовать понятия единица (1), ИСТИНА (TRUE) и ВЫСОКИЙ УРОВЕНЬ СИГНАЛА (HIGH) как синонимы. Аналогичным образом мы будем использовать ноль (0), ЛОЖЬ (FALSE) и НИЗКИЙ УРОВЕНЬ СИГНАЛА (LOW) как взаимозаменяемые термины.



Джордж Буль
1815–1864

Джордж Буль родился в семье небогатого ремесленника. Родители Джорджа не могли оплатить его формального образования, поэтому он осваивал математику самоучкой. Несмотря на это, Булю удалось стать преподавателем Королевского колледжа в Ирландии. В 1854 году Джордж Буль написал свою работу «Исследование законов мышления», которая впервые ввела в научный оборот двоичные переменные, а также три основных логических оператора И, ИЛИ, НЕ (AND, OR, NOT). (Портрет любезно предоставлен Американским физическим институтом.)

Преимущества *цифровой абстракции* заключаются в том, что разработчик цифровой системы может сосредоточиться исключительно на единицах и нулях, полностью игнорируя, каким образом логические переменные представлены на физическом уровне. Разработчика не волнует, представлены ли нули и единицы определенными значениями напряжения, вращающимися шестернями или уровнем гидравлической жидкости. Программист может продуктивно работать, не располагая детальной информацией об аппаратном обеспечении компьютера. Но понимание того, как работает это аппаратное обеспечение, позволяет программисту гораздо лучше оптимизировать программу для конкретного компьютера.

Как вы могли видеть выше, один-единственный бит не может передать большого количества информации. Поэтому в следующем разделе мы рассмотрим вопрос о том, каким образом набор битов можно использовать для представления десятичных чисел. В последующих главах мы также покажем, как группы битов могут представлять буквы и даже целую программу.

1.4. Системы счисления

Все мы привыкли работать с десятичными числами. Но в цифровых системах, построенных на единицах и нулях, использование двоичных или шестнадцатеричных чисел зачастую более удобно. В данном разделе мы рассмотрим системы счисления, использованные в этой книге.

1.4.1. Десятичная система счисления

Еще в начальной школе всех нас научили считать и выполнять различные арифметические операции в *десятичной* (decimal) системе счисления. Такая система использует десять арабских цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 – столько же, сколько у нас пальцев на руках. Числа больше 9 записываются в виде строки цифр. Причем цифра, находящаяся в каждой последующей позиции такой строки, начиная с крайней правой цифры, имеет «вес», в десять раз превышающий «вес» цифры, находящейся в предыдущей позиции. Именно поэтому десятичную систему счисления называют *системой по основанию* (base) 10. Справа налево «вес» каждой позиции увеличивается следующим образом: 1, 10, 100, 1000 и т. д. Позицию, которую цифра занимает в строке десятичного числа, называют разрядом, или декадой.

Чтобы избежать недоразумений при одновременной работе с более чем одной системой счисления, основание системы обычно указывается путем добавления цифры позади и чуть ниже основного числа: 9742_{10}. **Рисунок 1.4** показывает, для примера, как десятичное число 9742_{10} может быть записано в виде суммы цифр, составляющих это число, умноженных на «вес» разряда, соответствующего каждой конкретной цифре.

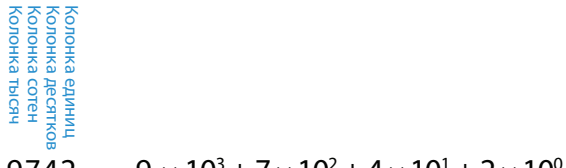


Рис. 1.4 Представление десятичного числа

N-разрядное десятичное число может представлять одну из 10^N цифровых комбинаций: 0, 1, 2, 3, ... $10^N - 1$. Это называется диапазоном *N*-разрядного числа. Десятичное число, состоящее из трех цифр (разрядов), например, представляет одну из 1000 возможных цифровых комбинаций в диапазоне от 0 до 999.

1.4.2. Двоичная система счисления

Одиночный бит может принимать одно из двух значений, 0 или 1. Несколько битов, соединенных в одной строке, образуют *двоичное* (binary) число. Каждая последующая позиция в двоичной строке имеет вдвое больший «вес», чем предыдущая позиция, так что двоичная система счисления – это система по основанию 2. В двоичном числе «вес» каждой позиции увеличивается (так же, как и в десятичном – справа налево) следующим образом: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16 384, 32 768, 65 536 и т. д. Работая с двоичными числами, очень полезно для экономии времени запомнить значения степеней двойки до 2^{16}.

Произвольное *N*-разрядное двоичное число может представлять одну из 2^N цифровых комбинаций: 0, 1, 2, 3, ... $2^N - 1$. В **табл. 1.1** собраны 1-битные, 2-битные, 3-битные и 4-битные двоичные числа и их десятичные эквиваленты.

Пример 1.1 ПРЕОБРАЗОВАНИЕ ЧИСЕЛ ИЗ ДВОИЧНОЙ СИСТЕМЫ СЧИСЛЕНИЯ В ДЕСЯТИЧНУЮ

Преобразовать двоичное число 10110_2 в десятичное.

Решение Необходимые преобразования представлены на **рис. 1.5**.

$$10110_2 = 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0 = 22_{10}$$

Одна шестнадцать · Нет восемь · Одна четыре · Одна двойка · Нет единиц

Рис. 1.5 Преобразование двоичного числа в десятичное число

Таблица 1.1 Таблица двоичных чисел и их десятичный эквивалент

1-битные двоичные числа	2-битные двоичные числа	3-битные двоичные числа	4-битные двоичные числа	Десятичные эквиваленты
0	00	000	0000	0
1	01	001	0001	1
	10	010	0010	2
	11	011	0011	3
		100	0100	4
		101	0101	5
		110	0110	6
		111	0111	7
			1000	8
			1001	9
			1010	10
			1011	11
			1100	12
			1101	13
			1110	14
			1111	15

Пример 1.2 ПРЕОБРАЗОВАНИЕ ЧИСЕЛ ИЗ ДЕСЯТИЧНОЙ СИСТЕМЫ СЧИСЛЕНИЯ В ДВОИЧНУЮ

Преобразовать десятичное число 84_{10} в двоичное.

Решение Определите, что должно стоять в каждой позиции двоичного результата: 1 или 0. Вы можете делать это, начиная с левой или правой позиции.

Если начать слева, найдите наибольшую степень 2, меньшую или равную заданному числу (в примере такая степень – это 64). 84 > 64, поэтому ставим 1 в позиции, соответствующей 64. Остается 84 – 64 = 20, 20 < 32, так что в позиции 32 надо поставить 0, 20 > 16, поэтому в позиции 16 ставим 1. Остается 20 – 16 = 4. 4 < 8, поэтому 0 в позиции 8. 4 ≥ 4 – ставим 1 в позицию 4. 4 – 4 = 0, поэтому будут 0 в позициях 2 и 1. Собрав все вместе, получаем $84_{10} = 1010100_2$.

Если начать справа, будем последовательно делить исходное число на 2. Остаток идет в очередную позицию. 84/2 = 42, поэтому 0 в самой правой позиции. 42/2 = 21, 0 во вторую позицию. 21/2 = 10, остаток 1 идет в позицию, соот-

ветствующую 4. $10/2 = 5$, поэтому 0 в позицию, соответствующую 8. $5/2 = 2$, остаток 1 в позицию 16. $2/2 = 1$, 0 в 32 позицию. Наконец, $1/2 = 0$ с остатком 1, который идет в позицию 64. Снова $84_{10} = 1010100_2$.

1.4.3. Шестнадцатеричная система счисления

Использование длинных двоичных чисел для записи и выполнения математических расчетов на бумаге утомительно и чревато ошибками. При этом длинное двоичное число можно разбить на группы по четыре бита, каждая из которых представляет одну из $2^4 = 16$ цифровых комбинаций. Именно поэтому зачастую бывает удобнее использовать для работы систему счисления по основанию 16, называемую *шестнадцатеричной* (hexadecimal). Для записи шестнадцатеричных чисел используются цифры от 0 до 9 и буквы от A до F, как показано в **табл. 1.2**. В шестнадцатеричном числе «вес» каждой позиции меняется следующим образом: 1, 16, 16^2 (или 256), 16^3 (или 4096) и т. д.

Интересно, что термин *hexadecimal* (шестнадцатеричный) введен в научный обиход корпорацией IBM в 1963 году и является комбинацией греческого слова *hexi* (шесть) и латинского decem (десять). Правильнее было бы использовать латинское же слово *sexa* (шесть), но термин *sexadecimal* воспринимался бы несколько неоднозначно.

Таблица 1.2 Шестнадцатеричная система счисления

Шестнадцатеричная цифра	Десятичный эквивалент	Двоичный эквивалент
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Пример 1.3 ПРЕОБРАЗОВАНИЕ ШЕСТНАДЦАТЕРИЧНОГО ЧИСЛА В ДВОИЧНОЕ И ДЕСЯТИЧНОЕ

Преобразовать шестнадцатеричное число $2ED_{16}$ в двоичное и десятичное.

Решение Преобразование шестнадцатеричного числа в двоичное и обратно – очень простое, так как каждая шестнадцатеричная цифра прямо соответствует

4-разрядному двоичному числу. $2_{16} = 0010_2$, $E_{16} = 1110_2$ и $D_{16} = 1101_2$, так что $2ED_{16} = 001011101101_2$. Преобразование в десятичную систему счисления требует выполнения арифметических операций, показанных, показанной на **рис. 1.6**.

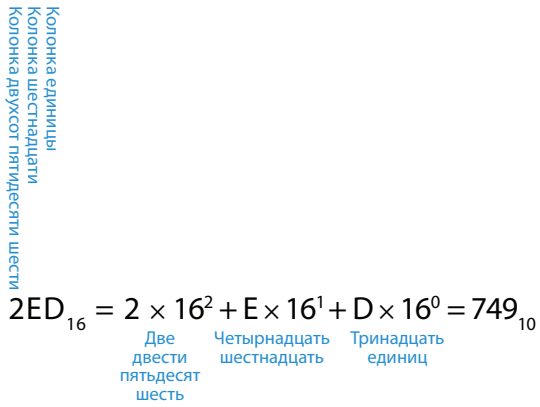


Рис. 1.6 Преобразование шестнадцатеричного числа в десятичное число

Пример 1.4 ПРЕОБРАЗОВАНИЕ ДВОИЧНОГО ЧИСЛА В ШЕСТНАДЦАТЕРИЧНОЕ

Преобразовать двоичное число 1111010_2 в шестнадцатеричное.

Решение Повторим еще раз, это просто. Начинаем справа. 4 наименее значимых бита $1010_2 = A_{16}$. Следующие биты $111_2 = 7_{16}$. Отсюда $1111010_2 = 7A_{16}$.

Пример 1.5 ПРЕОБРАЗОВАНИЕ ДЕСЯТИЧНОГО ЧИСЛА В ШЕСТАНДЦАТЕРИЧНОЕ И ДВОИЧНОЕ

Преобразовать десятичное число 333_{10} в шестнадцатеричное и двоичное.

Решение Как и в случае преобразования десятичного числа в двоичное, можно начать как слева, так и справа.

Если начать слева, найдите наибольшую степень шестнадцати, меньшую или равную заданному числу (в нашем случае это $16^2 = 256$). Число 256 содержится в числе 333 только один раз, поэтому в позицию с «весом» 256 мы записываем единицу. Остается число $333 - 256 = 77$. Число 16 содержится в числе 77 четыре раза, поэтому в позицию с «весом» 16 записываем четверку. Остается $77 - 16 \times 4 = 13$. $13_{10} = D_{16}$, поэтому в позицию с «весом» 1 записываем цифру D. Итак, $333_{10} = 14D_{16}$, это число легко преобразовать в двоичное, как мы показали в примере 1.3: $14D_{16} = 101001101_2$.

Если начинать справа, будем повторять деление на 16. Каждый раз остаток идет в очередную колонку. $333/16 = 20$ с остатком $13_{10} = D_{16}$, который идет в самую правую позицию. $20/16 = 1$ с остатком 4, который идет в позицию с «весом» 16. $1/16 = 0$ с остатком 1, который идет в позицию с «весом» 256. В результате опять получаем $14D_{16}$.

1.4.4. Байт, полубайт и «весь этот джаз»

Группа из восьми бит называется *байт* (byte). Байт представляет $2^8 = 256$ цифровых комбинаций. Размер модулей, сохраненных в памяти компьютера, обычно измеряется именно в байтах, а не битах.

Группа из четырех бит (половина байта) называется *полубайт* (nibble). Полубайт представляет $2^4 = 16$ цифровых комбинаций. Одна шестнадцатеричная цифра занимает один полубайт, а две шестнадцатеричные цифры – один байт. В настоящее время полубайты уже не находят широкого применения, но этот термин все же стоит знать, да и звучит он забавно (в англ. языке *nibble* означает откусывать что-либо маленькими кусочками).

Микропроцессор обрабатывает данные не целиком, а небольшими блоками, называемыми словами. Размер *слова* (word) не является величиной, установленной раз и навсегда, а определяется архитектурой каждого конкретного микропроцессора. На момент написания этой главы (в 2012 году) абсолютное большинство компьютеров использовало 64-битные процессоры. Такие процессоры обрабатывают информацию блоками (словами) длиной 64 бита. А еще не так давно верхом совершенства считались компьютеры, обрабатывающие информацию словами длиной 32 бита. Интересно, что и сегодня наиболее простые микропроцессоры и особенно те, что управляют работой таких бытовых устройств, как, например, тостеры или микроволновые печи, используют слова длиной 16 бит или даже 8 бит.

Если подходить абсолютно строго к терминологии, то микропроцессором называется такой процессор, все элементы которого размещаются на одной микросхеме. До 70-х годов XX века полупроводниковая технология не позволяла разместить процессор целиком на одной микросхеме, поэтому процессоры мощных компьютеров представляли собой набор плат с довольно большим количеством различных микросхем на них. Компания Intel в 1971 году представила первый 4-битный микропроцессор, получивший в качестве названия номер 4004. В наши дни даже самые передовые суперкомпьютеры построены на микропроцессорах, поэтому в этой книге мы будем считать «микропроцессор» и «процессор» тождественными понятиями и использовать оба этих термина как синонимы.

В рамках одной группы битов конечный бит, находящийся на одном конце этой группы (обычно правом), называется *наименее значимым битом* (least significant bit, LSB), или просто младшим битом, а бит на другом конце группы называется *наиболее значимым битом* (most significant bit, MSB), или *старшим битом*. **Рисунок 1.7 (a)** демонстрирует наименее и наиболее значимые биты в случае 6-битного двоичного числа. Аналогичным образом внутри одного слова можно выделить *наименее значимый байт* (least significant byte, LSB), или младший байт, и *наиболее значимый байт* (most significant byte, MSB), или старший байт. **Рисунок 1.7 (b)** показывает, как это делается в случае 4-байтного числа, записанного восемью шестнадцатеричными цифрами.

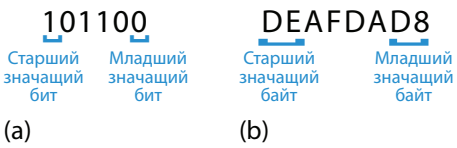


Рис. 1.7 Наименее и наиболее значимые биты и байты

В силу удачного совпадения $2^{10} = 1024 \approx 10^3$. Этот факт позволяет нам использовать приставку *кило* (греческое название тысячи) для сокращенного обозначения 2^{10}. Например, 2^{10} байт – это один килобайт (1 КБ). Подобным же образом *мега* (греческое название миллиона) обозначает $2^{20} \approx 10^6$, а *гига* (греческое название миллиарда) указывает на $2^{30} \approx 10^9$. Зная, что $2^{10} \approx 1$ тысяча, $2^{20} \approx 1$ миллион, $2^{30} \approx 1$ миллиард и помня значения степеней двойки до 2^9 включительно, будет легко приблизительно рассчитать в уме любую другую степень двух.

Пример 1.6 ОЦЕНКА СТЕПЕНЕЙ ДВОЙКИ

Найдите приблизительное значение 2^{24} без использования калькулятора.

Решение Представьте экспоненту как число, кратное десяти, и остаток.

$2^{24} = 2^{20} \times 2^4$, $2^{20} \approx 1$ миллион. $2^4 = 16$. Итак, $2^{24} \approx 16$ миллионов. На самом деле $2^{24} = 16\,777\,216$, но 16 миллионов – достаточно хорошее приближение для маркетинговых целей.

Так же как 1024 байта называют *килобайтом* (КБ), 1024 бита называют *килобитом* (Кб или кбит). Аналогичным образом МБ, Мб, ГБ и Гб используются для сокращенного обозначения миллиона и миллиарда байт и бит. Размеры элементов памяти обычно измеряются в байтах. А вот скорость передачи данных измеряется в битах в секунду. Максимальная скорость передачи данных телефонным модемом, например, составляет 56 килобит в секунду.

1.4.5. Сложение двоичных чисел

Сложение двоичных чисел производится так же, как и сложение десятичных, с той лишь разницей, что двоичное сложение выполнить гораздо проще (**рис. 1.8**). Как и при сложении десятичных чисел, если сумма двух чисел превышает значение, помещающееся в один разряд, мы переносим 1 в следующий разряд. На **рис. 1.8** для сравнения показано сложение десятичных и двоичных чисел. В крайней правой колонке на **рис. 1.8 (a)** складываются числа 7 и 9. Сумма 7 + 9 = 16, что превышает 9, а значит, больше того, что может вместить один десятичный разряд. Поэтому мы записываем в первый разряд 6 (первая колонка) и переносим 10 в следующий разряд (вторая колонка) как 1. Аналогичным же образом при сложении двоичных чисел, если сумма двух чисел превышает 1, мы переносим 2 в следующий разряд как 1. В правой колонке на **рис. 1.8 (b)**, например, сумма $1 + 1 = 2_{10} = 10_2$, что не может уместиться в одном двоичном разряде. Поэтому мы записываем 0 в первом разряде (первая колонка) и 1 в следующем разряде (вторая колонка). Во второй колонке опять складываются 1 и 1 и еще добавляется 1, перенесенная сюда после сложения чисел в первой колонке. Сумма $1 + 1 + 1 = 3_{10} = 11_2$.

Мы записываем 1 в первый разряд (вторая колонка) и снова добавляем 1 в следующий разряд (третья колонка). По очевидной причине бит, добавленный в соседний разряд (колонку), называется *битом переноса* (carry bit).

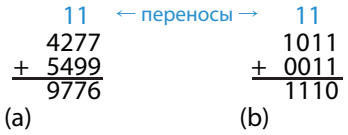


Рис. 1.8 Примеры сложения с переносом: (a) десятичное, (b) двоичное

Пример 1.7 ДВОИЧНОЕ СЛОЖЕНИЕ

Вычислить $0111_2 + 0101_2$.

Решение На **рис. 1.9** показано, что сумма равна 1100_2. Переносы выделены синим цветом. Мы можем проверить нашу работу, повторив вычисления в десятичной системе счисления. $0111_2 = 7_{10}$, $0101_2 = 5_{10}$. Сумма равна $12_{10} = 1100_2$.

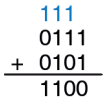


Рис. 1.9 Пример двоичного сложения

Цифровые системы обычно оперируют числами с заранее определенным и фиксированным количеством разрядов. Ситуацию, когда результат сложения превышает выделенное для него количество разрядов, называют *переполнением* (overflow). Четырехбитная ячейка памяти, например, может сохранять значения в диапазоне [0, 15]. Такая ячейка переполняется, если результат сложения превышает число 15. В этом случае дополнительный пятый бит отбрасывается, а результат, оставшийся в четырех битах, будет ошибочным. Переполнение можно обнаружить, если следить за переносом бита из наиболее значимого разряда двоичного числа (**рис. 1.8**), из наиболее левой колонки.

Пример 1.8 СЛОЖЕНИЕ С ПЕРЕПОЛНЕНИЕМ

Вычислить $1101_2 + 0101_2$. Будет ли переполнение?

Решение На **рис. 1.10** показано, что сумма равна 10010_2. Результат выходит за границы четырехбитового двоичного числа. Если его нужно запомнить в 4 битах, наиболее значимый бит пропадет, оставив некорректный результат 0010_2. Если вычисления производятся с числами с пятью или более битами, результат 10010_2 будет корректным.

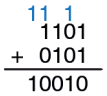


Рис. 1.10 Пример двоичного сложения с переполнением

1.4.6. Знак двоичных чисел

До сих пор мы рассматривали двоичные числа *без знака* (unsigned) – то есть только положительные числа. Часто для вычислений требуются как положительные, так и отрицательные числа, а это значит, что для знака двоичного числа нам потребуется дополнительный разряд. Существует

несколько способов представления двоичных чисел *со знаком* (signed). Наиболее широко применяются два: *прямой код* (Sign / Magnitude) и *дополнительный код* (Two's Complement).

Прямой код

Представление отрицательных двоичных с использованием прямого кода интуитивно покажется вам наиболее привлекательным, поскольку совпадает с привычным способом записи отрицательных чисел, когда сначала идет знак минус, а затем абсолютное значение числа. Двоичное число, состоящее из N бит и записанное в прямом коде, использует наиболее значимый бит для знака, а остальные $N - 1$ бит для записи абсолютного значения этого числа. Если наиболее значимый бит 0, то число положительное. Если наиболее значимый бит 1, то число отрицательное.

Пример 1.9 ПРЕДСТАВЛЕНИЕ ЧИСЕЛ В ПРЯМОМ КОДЕ

Запишите числа 5 и −5 как четырехбитовые числа в прямом коде.

Решение Оба числа имеют абсолютную величину $5_{10} = 101_2$. Таким образом, $5_{10} = 0101_2$ и $-5_{10} = 1101_2$.

К сожалению, стандартный способ сложения не работает в случае двоичных чисел со знаком, записанных в прямом коде. Например, складывая $-5_{10} + 5_{10}$ привычным способом, получаем $1101_2 + 0101_2 = 10010_2$. Что, естественно, является полным абсурдом.

Двоичная переменная длиной N бит в прямом коде может представлять число в диапазоне $[-2^{N-1} + 1, 2^{N-1} - 1]$.

Другой несколько странной особенностью прямого кода является наличие +0 и −0, причем оба этих числа соответствуют одному нулю. Нетрудно предположить, что представление одной и той же величины двумя различными способами чревато ошибками.

Дополнительный код

Двоичные числа, записанные с использованием дополнительного кода, и двоичные числа без знака идентичны, за исключением того, что в случае дополнительного кода вес наиболее значимого бита -2^{N-1} вместо 2^{N-1}, как в случае двоичного числа без знака. Дополнительный код гарантирует однозначное представление нуля, допускает сложение чисел по привычной схеме, а значит, избавлен от недостатков прямого кода.

В случае дополнительного кода нулевое значение представлено нулями во всех разрядах двоичного числа: $00...000_2$. Максимальное положительное значение представлено нулем в наиболее значимом разряде и единицами во всех других разрядах двоичного числа: $01...111_2 = 2^{N-1} - 1$. Максимальное отрицательное значение содержит единицу в наиболее

значимом разряде и нули во всех остальных разрядах: $10...000_2 = -2^{N-1}$. Отрицательная единица представлена единицами во всех разрядах двоичного числа: $11...111_2$.

Обратите внимание на то, что наиболее значимый разряд у всех положительных чисел – это «0», в то время как у отрицательных чисел – это «1», то есть наиболее значимый бит дополнительного кода можно рассматривать как аналог знакового бита прямого кода. Но на этом сходство кончается, поскольку остальные биты дополнительного кода интерпретируются не так, как биты прямого кода.

В случае дополнительного кода знак отрицательного двоичного числа изменяется на противоположный путем выполнения специальной операции, называемой *в дополнительном коде* (taking the two's complement). Суть этой операции заключается в том, что инвертируются все биты этого числа, а затем к значению наименее значимого бита прибавляется 1. Подобная операция позволяет найти двоичное представление отрицательного числа или определить его абсолютное значение.

Пример 1.10 ПРЕДСТАВЛЕНИЕ ОТРИЦАТЕЛЬНЫХ ЧИСЕЛ В ДОПОЛНИТЕЛЬНОМ КОДЕ

Найти представление -2_{10} как 4-битового числа в дополнительном коде.

Решение Начните с $+2_{10} = 0010_2$. Для получения -2_{10} инвертируйте биты и добавьте единицу. Инвертируя 0010_2, получим 1101_2. $1101_2 + 1 = 1110_2$. Итак, -2_{10} равно 1110_2.

Пример 1.11 ЗНАЧЕНИЕ ОТРИЦАТЕЛЬНЫХ ЧИСЕЛ В ДОПОЛНИТЕЛЬНОМ КОДЕ

Найти десятичное значение числа 1001_2 в дополнительном коде.

Решение Число 1001_2 имеет старшую 1, поэтому оно должно быть отрицательным. Чтобы найти его модуль, инвертируем все биты и добавляем 1. Инвертируя 1001_2, получим 0110_2. $0110_2 + 1 = 0111_2 = 7_{10}$. Отсюда $1001_2 = -7_{10}$.

Неоспоримым преимуществом дополнительного кода является то, что привычный способ сложения работает как в случае положительных, так и отрицательных чисел. Напомним, что при сложении N-битных чисел N-й бит (т. е. $N + 1$ бит результата) не переносится.

(Фото: ESA/CNES/ ARIANESPACE-Service Optique CS6)

Ракета Ариан-5 ценой 7 млрд долларов, запущенная 4 июня 1996 года, отклонилась от курса и разрушилась через 40 секунд после запуска. Отказ был вызван тем, что в бортовом компьютере произошло переполнение 16-разрядных регистров, после чего компьютер вышел из строя.

Программное обеспечение Ариан-5 было тщательно протестировано, но на ракете Ариан-4. Но новая ракета имела двигатели с более высокими скоростными параметрами, которые, будучи переданными бортовому компьютеру, и вызвали переполнение регистров.

Пример 1.12 СЛОЖЕНИЕ ЧИСЕЛ, ПРЕДСТАВЛЕННЫХ В ДОПОЛНИТЕЛЬНОМ КОДЕ

Вычислить (a) $-2_{10} + 1_{10}$ и (b) $-7_{10} + 7_{10}$ с помощью чисел в дополнительном коде.

Решение

(a) $-2_{10} + 1_{10} = 1110_2 + 0001_2 = 1111_2 = -1_{10}$.

(b) $-7_{10} + 7_{10} = 1001_2 + 0111_2 = 10000_2$. Пятый бит отбрасывается, оставляя правильный 4-битовый результат 0000_2.

Вычитание одного двоичного числа из другого осуществляется путем преобразования вычитаемого в дополнительный код и последующего его сложения с уменьшаемым.

Пример 1.13 ВЫЧИТАНИЕ ЧИСЕЛ В ДОПОЛНИТЕЛЬНОМ КОДЕ

Вычислить (a) $5_{10} - 3_{10}$ и (b) $3_{10} - 5_{10}$, используя 4-разрядные числа в дополнительном коде.

Решение

(a) $3_{10} = 0011_2$. Вычисляя его дополнительный код, получим $-3_{10} = 1101_2$. Теперь сложим $5_{10} + (-3_{10}) = 0101_2 + 1101_2 = 0010_2 = 2_{10}$. Отметим, что перенос из наиболее значимой позиции сбрасывается, поскольку результат записывается в четырех битах.

(b) Вычисляя дополнительный код от 5_{10}, получим $-5_{10} = 1011_2$. Теперь сложим $3_{10} + (-5_{10}) = 0011_2 + 1011_2 = 1110_2 = -2_{10}$.

Представление нуля в дополнительном коде также производится путем инвертирования всех битов (это дает $11...111_2$) и последующим прибавлением 1, что делает значения всех битов равными 0. При этом перенос наиболее значимого бита игнорируется. В результате нулевое значение всегда представлено набором только нулевых битов. В отличие от прямого кода дополнительный код не имеет отрицательного нуля. Ноль всегда считается положительным числом, так как его знаковый бит всегда 0.

Так же, как и двоичное число без знака, произвольное N-битное число, записанное в дополнительном коде, может принимать одно из 2^N возможных значений. Но весь этот диапазон разделен между положительным и отрицательным числами. Например, 4-битное двоичное число без знака может принимать 16 значений от 0 до 15. В случае дополнительного кода 4-битное число также принимает 16 значений, но уже от -8 до 7. В общем случае диапазон N-битного числа, записанного в дополнительном коде, охватывает $[-2^{N-1}, 2^{N-1} - 1]$. Легко понять, почему в отрицательном диапазоне оказалось на одно значение больше, чем в положительном, – в дополнительном коде отсутствует отрицательный ноль.

Максимальное отрицательное число, которое можно записать, используя дополнительный код $10...000_2 = -2^{N-1}$, иногда называют *странным числом* (weird number). Чтобы представить это число в дополнительном коде, инвертируем все его биты (это даст нам $01...111_2$), прибавим 1 и получим в результате $10...000_2$ – опять это же самое «странное» число. То есть это единственное отрицательное число, которое не имеет положительной пары.

В случае дополнительного кода сложение двух положительных или отрицательных N-битовых чисел может привести к переполнению, если результат будет больше, чем $2^{N-1} - 1$, или меньше, чем -2^{N-1}. Сложение положительного и отрицательного чисел, напротив, никогда не приводит к переполнению. В отличие от двоичного числа без знака перенос наиболее значимого бита не является признаком переполнения. Вместо этого индикатором переполнения является ситуация, когда после сложения двух чисел с одинаковым знаком знаковый бит суммы не совпадает со знаковыми битами слагаемых.

Пример 1.14 СЛОЖЕНИЕ ЧИСЕЛ В ДОПОЛНИТЕЛЬНОМ КОДЕ С ПЕРЕПОЛНЕНИЕМ

Вычислить $4_{10} + 5_{10}$, используя четырехбитные числа в дополнительном коде. Произойдет ли переполнение?

Решение $4_{10} + 5_{10} = 0100_2 + 0101_2 = 1001_2 = -7_{10}$. Результат не помещается в диапазон положительных четырехбитных чисел в дополнительном коде, оказываясь отрицательным. Если бы вычисление выполнялось с пятью или более битами, результат был бы $01001_2 = 9_{10}$, что правильно.

В случае необходимости увеличения количества битов произвольного числа, записанного в дополнительном коде, значение знакового бита должно быть скопировано в наиболее значимые разряды модифицированного числа. Эта операция называется *знаковым расширением* (sign extension). Например, числа 3 и –3 записываются в 4-битном дополнительном коде как 0011 и 1101 соответственно. Если мы увеличиваем число разрядов до семи бит, мы должны скопировать знаковый бит в три наиболее значимых бита модифицированного числа, что дает 0000011 и 1111101.

Сравнение способов представления двоичных чисел

Три наиболее часто использующихся на практике способа представления двоичных чисел – это двоичные числа без знака, прямой код и дополнительный код. **Таблица 1.3** сравнивает диапазон N-битных чисел для каждого из этих трех способов. Преимущества дополнительного кода заключаются в том, что его можно использовать для представления

как положительных, так и отрицательных целых чисел, а привычный способ сложения работает для всех чисел, представленных в дополнительном коде. Вычитание осуществляется путем преобразования вычитаемого в отрицательное число и последующего сложения с уменьшаемым. В дальнейшем в этой книге, если не указано иное, предполагается, что все двоичные числа представлены в дополнительном коде.

Таблица 1.3 Диапазон *N*-битных чисел

Система	Диапазон
Двоичные числа без знака	$[0, 2^N - 1]$
Прямой код	$[-2^{N-1} + 1, 2^{N-1} - 1]$
Дополнительный код	$[-2^{N-1}, 2^{N-1} - 1]$

На **рис. 1.11** изображена десятичная числовая шкала с соответствующими десятичными и 4-битными двоичными числами, представленными тремя вышеперечисленными способами. Двоичные числа без знака находятся в диапазоне [0, 15] и располагаются в обычном порядке. 4-битные двоичные числа, представленные в дополнительном коде, занимают диапазон [−8, 7]. Причем положительные числа [0, 7] используют точно такую же кодировку, как и двоичные числа без знака. Отрицательные же числа [−8, −1] кодируются таким образом, что наибольшее двоичное значение каждого такого числа без знака представляет число, наиболее близкое к 0. Обратите внимание на то, что «странное число» 1000 соответствует десятичному значению −8 и не имеет положительной пары. Числа, представленные в прямом коде, занимают диапазон [−7, 7]. При этом наиболее значимый бит является знаковым. Положительные числа [0, 7] используют такую же кодировку, как и двоичные числа без знака. Отрицательные числа симметричны положительным, с той лишь разницей, что их знаковый бит имеет значение 1. Ноль представлен двумя значениями 0000 и 1000. В результате того, что два числа соответствуют одному нулю, любое произвольное N-разрядное двоичное число в прямом коде может представлять только $2^N - 1$ целых числа.

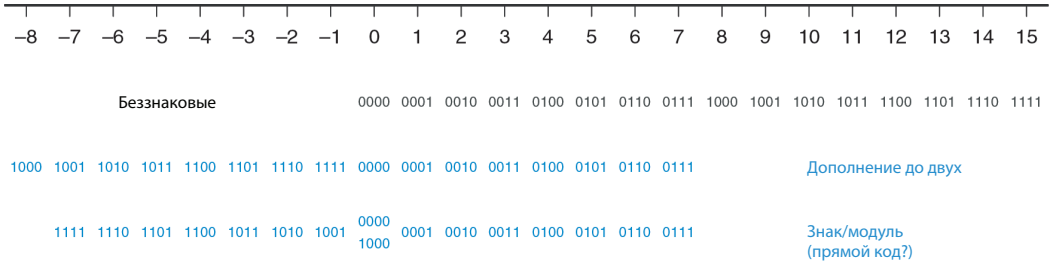


Рис. 1.11 Числовая шкала и 4-битовое двоичное кодирование