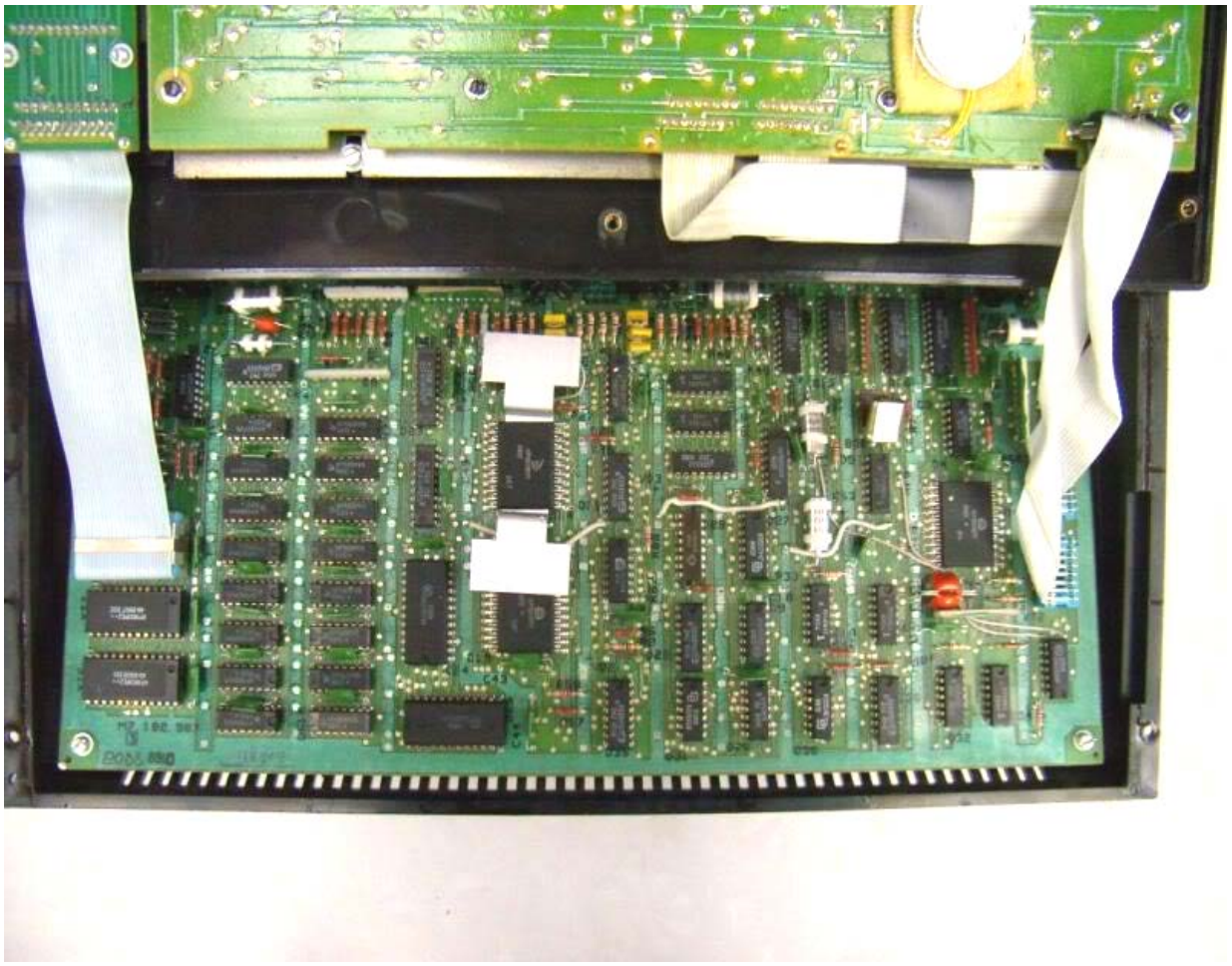
Давайте разберём какой-нибудь старый компьютер, годов этак семидесятых или восьмидесятых. Мы там увидим кучу микросхем, припаянных к плате. А по плате между ними пролегают тысячи дорожек, соединяющих ножки этих самых микросхем.



Самое интересное заключается в том, что все эти микросхемы по своей сути являются достаточно однородными вещами. Они всего лишь выполняют какие-либо логические функции. Какая-то делает операцию И (на C++ это операция &&), какая-то – ИЛИ, какая-то – НЕ, есть какие-то ещё функции (сложение, сравнение, ИСКЛЮЧАЮШЕЕ ИЛИ и т.п). Но суть одна. Если нам на языке программирования надо выполнить какое-то действие, мы пишем:

```
a = b + c * d
```

или, скажем

```
if ((a+b>2) && (c-d == 4))
```

А если мы сидим в далёком 84-м году и не пишем программу для компьютера, а делаем схему, которая реализует что-то подобное, то для каждого оператора (+, >, &&, == и т.п.) нам придётся собрать какую-то схемку из нескольких элементов, каждый из которых прописан в каком-либо уголке какой-либо микросхемы (детали – не для этой статьи, если надо – я лучше лекцию устно прочитаю). Разумеется, уже тогда большие зарубежные фирмы под свои нужны заказывали что-то более компактное. Например, ZX Spectrum, в который наши любители паяли по 42 микросхемы, в оригинале содержал их намного меньше (процессор, память и один чип, который делал всё остальное). Но эти Большие Интегральные Схемы (БИС) были узко специализированы. Если она сделана для

ZX Spectrum, то больше никуда её не поставишь. И в ~~колбасном магазине~~ «Радиодеталях» не купишь. Ибо их тысячи видов, не держать же все на складе. Поэтому если ты – не большая корпорация, готовая заказать сначала разработку БИС, а потом – производство пары миллионов штук, то судьба тебе всё делать на мелкой логике (логика-то мелкая, а платы на ней получаются здоровенные).

И вдруг появился луч света в тёмном царстве. Его звали (да простит нас Белинский) не Катериной, а ПЛМ (Программируемая Логическая Матрица). За границей же его прозвали PLD (Programmable Logic Device). Кто учился на ПОВТе, те слушали лекции по схемотехнике. И там нам много раз говорили непонятное слово «Минтерм». Лично я на лекциях не мог понять, зачем он нужен, если на мелкой логике я чисто умозрительной оптимизацией сделаю намного компактнее, чем с его помощью… Но прелесть этих минтермов в том, что если отбросить оптимизацию, то на них можно сделать всё, что только вздумается. А в этом нам поможет универсальный элемент И-ИЛИ-НЕ. Он состоит из нескольких элементов И, которые потом объединяются по ИЛИ, а затем – если надо, ещё и делают НЕ (кто привык к Си, читаем &&, || и !). Доказательство этого факта отложим на лекцию (если кто согласится её прослушать), а пока поверьте на слово.

И вот разработчики ПЛМ взяли этот элемент и чуточку доработали его напильником. Они добавили на всех связях пережигаемые перемычки.

Представьте себе строчку на Си:

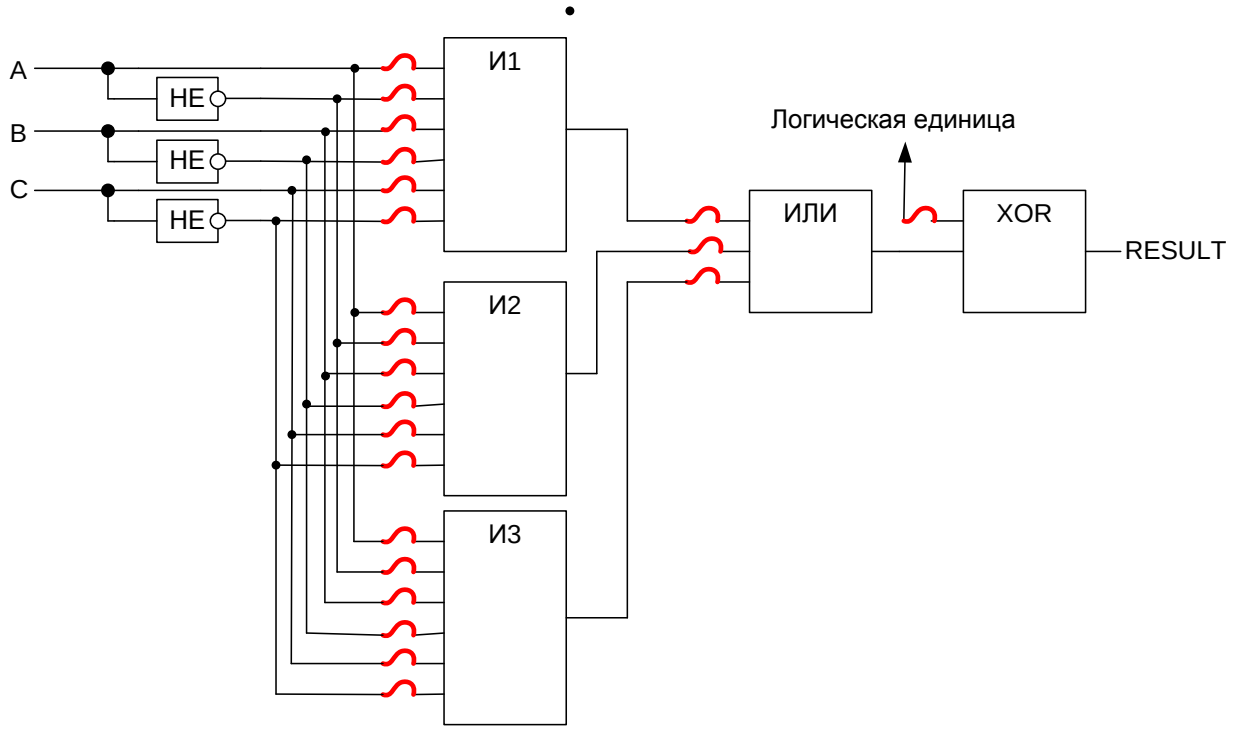
```
if ((A&&B&&C&&D)||(E&&F&&G&&H))
```

в которой можно оставить все записи, а можно – какие-то из них просто вычеркнуть. Допустим, мы взяли и пережгли перемычки на входе &&C, &&F, &&G. Получили выражение:

```
if ((A&&B&&D)||(E&&H))
```

Ну, и так далее. В общем, нам дают максимально возможную программу, а мы из неё вычёркиваем лишнее. Сегодня пришёл один Заказчик – мы вычеркнули одно, завтра другой – другое, послезавтра – третье и так далее.

То же самое и с ПЛМ. После того, как я написал программистский аналог, позволю себе привести структуру ПЛМ с точки зрения схемотехника (для простоты я ограничусь тремя входами A, B и C, а также тремя элементами И, но реально их НАМНОГО больше). Красными на рисунке обозначены пережигаемые перемычки, именно в них вся соль ПЛМ. Если перемычка имеется – на вход элемента подаётся сигнал, если её заранее пережгли (при помощи программатора) – логический ноль. С завода микросхема приходит со всеми перемычками, жжём же мы их по мере надобности.



Мне довелось ощутить всю прелесть ПЛМ, когда я в 1998 году делал аналог платы PC-3000 (знаменитый ремонтный комплекс для накопителей). Первая версия была забабахана мною на мелкой логике. Там была куча микросхем, спаянных проводочками. По мере понимания сути защиты PC-3000 приходилось всё перепаивать, а периодически – добавлять новые элементы.

Потом я увидел в продаже **ЕЁ. ОНА** стоила дешевле, чем стаканчик мороженого, а уж те чудеса, которые она творила в… Извините, но панельки для микросхем на жаргоне называются кроватками… В общем, я влюбился в программируемую логику всерьёз и надолго (уж 11 лет, как любовь не проходит). Даже если учесть, что в те времена не было флэшек, а перемычки пережигались на века – не беда. Обратите внимание, что на все входы исходно подан сигнал A и !A. Значит результат – FALSE. Так? Пусть мы использовали элементы И1 и И2. Потом при отладке устройства выяснено, что функция И2 сделана с ошибкой. Раньше пришлось бы браться за паяльник и корёжить схему, а теперь – мы просто отжигаем И2 от элемента ИЛИ, выводя его из игры, а правильную функцию прожигаем в И3. И так далее. Романтика! За всё время опытов с той самой платой, я не использовал даже четверти ёмкости ПЛМки! И всё это – за три тогдашних рубля.

Шли годы. Технологии развивались. На смену пережигаемым перемычкам пришли FLASH-технологии, а степень интеграции стала такой, что в один корпус стало возможно уместить не одну, а несколько подобных структур. Такие микросхемы стали называться CPLD (русского названия у них нет, так как Советская промышленность к тому времени уже была мертва, а Российская до сих пор не родилась). CPLD – это Complex Programmable Logic Device. Разумеется, в них было натолкано ещё много чего полезного (триггеры, трассировочные ресурсы, прочее), о чём я не буду писать в статье (будем бороться со сном на рабочем месте, так что снотворных статей я писать не буду, если надо – это надо не одну, а несколько лекций рассказывать). А так – идея осталась прежняя. Имеются большие функции И-ИЛИ-НЕ, связи между которыми могут разрываться.

У такой конструкции есть масса достоинств, но зачастую она неоптимальна. Во-первых, функциями И-ИЛИ-НЕ можно описать всё, но смотрите выше про мои

недоумения на лекциях. В Москву через Мурманск тоже можно гарантированно добраться, но оптимальность всё же не та. Кроме того, если нам нужно сделать функцию от пяти аргументов, всё равно в микросхеме будут стоять, скажем, 32 входовые элементы И. Жуть! Поэтому, несмотря на то, что CPLD живут и процветают (в некоторых случаях только на них и получается сделать), была разработана другая группа программируемых логических устройств, в основе которых лежит ОЗУ.

Давайте подумаем, что такое память? Это некий массив. Ну, раз мы программисты, то вот нам массив

BYTE mem [0x10];

Индекс в массиве железячники называют адресом. Содержимое массива – данными. Давайте вспомним, что все числа в компьютерах – двоичные. Значит и адрес и данные внутри машины представлены в двоичном виде. А давайте-ка зарисуем массив с учётом этих знаний… Данные я заполнил от балды, зачем – поясню ниже.

Адрес	Двоичное представление адреса	Данные							
		Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
0x00	0000	1	1	0	1	0	1	1	1
0x01	0001	1	0	1	0	0	1	0	0
0x02	0010	0	0	1	1	0	0	0	0
0x03	0011	0	1	0	1	0	1	1	1
0x04	0100	1	0	1	1	0	0	1	0
…	…								

Итак, мы записали наш массив, используя двоичное представление как адреса, так и данных. А теперь внимание! Все, кто хоть раз минимизировал функции с помощью уже упоминавшихся минтермов, сразу заметят, что эта таблица очень напоминает таблицу истинности функции, ведь любая, самая наизаковыристая функция может быть представлена в виде таблицы истинности (подробности оставим на лекцию, если таковая состоится). Получается, что функцию можно описать с помощью небольшой ОЗУшки (так память называется), залив в неё таблицу истинности. И создатели FPGA пошли именно таким путём. Они берут именно небольшие ОЗУшки (у CPLD могут быть десятки входов, а у функций FPGA – строго четыре, не спрашивайте почему, но почему-то все ставят именно четыре, наверняка есть пятитомное доказательство того, что это число наиболее оптимально), на выходе каждой ОЗУшки ставят триггер, и прокладывают между этим безобразием программируемые связи. И получается устройство, которое называется FPGA - Field-Programmable Gate Array.

Коль скоро это устройство является схемотехнической поделкой, «прошивку» для него можно рисовать в чисто схемотехническом стиле. Но стал бы я рекомендовать эту вещь для Компании, где работает масса программистов? Нет! Для них можно вести разработку на языке программирования. В Европе популярен язык VHDL, в Штатах – Verilog.

Ну вот, вкратце, о структуре ПЛИС. А дальше я просто процитирую другой документ, который был уже разработан для отдела продаж. В нём раскрываются прикладные свойства ПЛИС.

ПЛИС глазами схемотехника

С точки зрения схемотехника, ПЛИС – это микросхема, в которой есть много-много-много деталей, которые лежат себе и ничего не делают. Загружая прошивку в ПЛИС, мы всего лишь соединяем эти детали между собой так, как нужно нам. В результате, получается схема. В классическом же варианте, нам бы пришлось соединять детали при помощи паяльника. А чтобы поменять схему – сначала убрать старые проводочки, потом – припаять новые. Здесь же, достаточно загрузить другую прошивку (2-3 секунды), и получится новая схема. Ну, и разработка ведётся при помощи «мышки»

ПЛИС глазами программиста

Как уже говорилось, ПЛИС можно программировать на языке программирования. Так а чем она тогда отличается от процессора?

Процессор выполняет программу по шагам. Взял команду, при исполнении она откуда-то засосала аргументы, выполнила действия, куда-то затолкала результат. Перешли к следующей команде. И так – по шагам. Медленно, но верно. Хотим быстрее – ну берём Core Duo. Хотим ещё быстрее – берём Core Quad. Хотим ещё быстрее – ждём более крутых процессоров. Как бы там ни было, программа в процессоре выполняется шаг за шагом. Все параллельные вычисления, вся многозадачность – это то же самое, что кино. В кино нам кажется, что всё движется, а на самом деле, просто наш глаз не видит, что это много-много статичных кадров. То же самое и с многозадачностью. Задачи выполняются по одной, просто они переключаются так часто, что мы не замечаем этого.

А раз ПЛИС – это много-много-много деталей, то они работают независимо друг от друга. Поэтому программа для ПЛИС исполняется истинно параллельно.

ПЛИС глазами продавца

Чтобы начать рассматривать ПЛИС глазами продавца, приведу старую шутку

Игра на вечер "ПОМОГАЕТ ПОСЛЕ ТЯЖЕЛОГО ТРУДОВОГО РАБОЧЕГО ДНЯ".

Для игры Вам понадобится: Человек (Вы сами), ванная с натянутыми над ней верёвками-лёсками для сушки белья, колготки.
Как играть:

1. Надеваете колготки попой на голову.
2. Залезаете в ванну.
3. Встаёте в ней в полный рост.
4. Ноги (колготочные, не свои) забрасываете на верёвки для сушки белья.
5. Медленно трогаетесь с места и делаете несколько шагов вперёд.
Вы - троллейбус!
6. Если при этом надеть лыжи - получается трамвай.
7. А если надеть лыжи и налить в ванну воды - получается речной трамвай.
8. А если в ванну с водой уронить включенный фен, то получится электричка.
9. А если выключить свет и одеть налобный фонарик, то Вы - метро.
Хорошего вечера !

А теперь – то же самое, но уже из жизни. По мере разработок для Cassel Aero, там накопилось достаточно много всякого оборудования, подобранного как нами, так и

Заказчиком. И это оборудование страшно дралось друг с другом, в силу своей разнородности. Так дралось, что Заказчик в прошлом году еженедельно звонил, иногда даже в 22 часа, приходилось рассказывать, как всё друг с другом согласовать. Жуть! Разумеется, было решено отойти от этой порочной практики и сделать что-то однородное, не такое универсальное, как каждый из компонентов, но зато идеально выполняющее функции в данном проекте. Кстати, об идеале. Часть требований, предъявляемых в этом проекте, на универсальной схеме не удавалось выполнить. Разумеется, Заказчик считал, что глючит наш софт, и единственный способ доказать, что это не так, был как раз в изготовлении правильного железа.

Но с другой стороны, у Заказчика нет дяди Рокфеллера. Поэтому когда ему была предложена раскладка по разработке, от ответа сильно пахло валерьянкой. Пришлось резко урезать сроки разработки, иначе был риск потерять проект вовсе. И тут я вспомнил, что для знакомых, в свободное от работы время, уже разработана плата на базе USB микроконтроллера и ПЛИС. Знакомые через неё читают микросхемы NAND Flash. Вроде, совсем не то, что надо, но мы же имеем дело с ПЛИС!

Поэтому Заказчику (в силу хороших с ним отношений) железо было подарено (минус 2-3 человеко-месяца). Была разработана только простенькая платка, которая преобразует нежные USB уровни (3 вольта) к суровым вертолётным (12-24 вольт), но платка – простая. Вся сложная схема, обеспечивающая работу кучи прибамбасов (COM порты с почти нулевым лагом – вот их-то как раз и не купить, везде лаг большой, управление двигателями, обслуживание лазерного дальномера и ещё куча финтифлюшек) были нарисованы чисто мышкой.

На обратном пути из командировки, я встречался со своим знакомым. Ему был нужен анализатор IDE накопителя. Ну нужен и всё тут. Не вопрос, в аэропорту, при помощи мышки, этот же самый блок (ну был он у меня с собой) превратился в анализатор, пришлось только кабель спаять.

А когда было срочно нужно подключить накопитель к ноутбуку, и, как на грех, не нашлось никакого переходника (ночь на дворе, где его купишь?), этот анализатор, при помощи одной мышиной силы, превратился в переходник USB->IDE.

Ну, и наконец, этой осенью, возникло подозрение, что в плате, преобразующей уровни, одна микросхема не обладает достаточной надёжностью. Немного подумав, мы решили, что её можно выкинуть вообще. Но чтобы её выкинуть, в классическом варианте пришлось бы перелопатить всю схему, докупив кучку других элементов. Но, к счастью, раз мы имели дело с ПЛИС, схема была перелопачена при помощи мышки, а в железе – пришлось сделать два пореза и бросить два проводочка.

Итого. ПЛИС позволяют сделать одно электрическое решение, а затем – применять его везде, где только можно (и даже где, казалось бы, нельзя), меняя функциональность железа до неузнаваемости.

Что такое ядро

Ядро – это аналог приложения для ЭВМ. Например, для Cassel Aero было разработано ядро COM порта, после чего шесть экземпляров ядра были помещены в ПЛИС. Кроме того, были разработаны ядра для поддержки лазерного дальномера (помещён один

экземпляр), для управления двигателями (помещено два экземпляра) и т.п. Таким образом, прошивка для ПЛИС состоит из ядер, соединённых между собой в работающую систему.

Цена ПЛИС

Ну, бывают ПЛИС за 45 тысяч долларов. Правда, если брать две, то цена уже будет 30 тысяч. Бывают ПЛИС за 1000 долларов. Но это – если мы для военных чего крутое будем делать. А так – пиковая цена ширпотребовской ПЛИС 50-100 долларов. Для Cassel Aero мы взяли ПЛИС за 18 долларов, и на текущий момент не задействовали и 50% её ёмкости. Вторая ПЛИС (модуль весов) для Cassel Aero стоит 1.5 доллара (правда, её возможности весьма скромны, но согласитесь, цена – тоже правильная, скоро мелкую логику, которая в десятки раз слабее, нельзя будет по такой цене купить).

А как выглядит ПЛИС?

ПЛИС в основном модуле для Cassel Aero

Рисунок 1. Основная плата Cassel Aero (я постарался сделать масштаб 1:1)

ПЛИС в модуле весов для Cassel Aero

Рисунок 2. Модуль весов (масштаб почти 1:1)