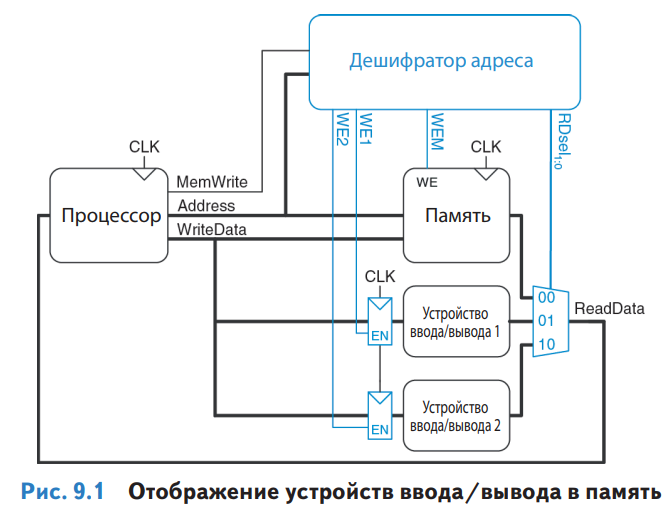


Процессор выставляет адреса на шину адреса (Address), идущую к подсистеме памяти. Для чтения управляющий сигнал записи (MemWrite) устанавливают в низкий уровень, а память возвращает данные по шине чтения данных (ReadData). Для записи MemWrite устанавливается в единицу, и процессор посылает данные в память по шине записи данных (WriteData).

Часть адресного пространства вместо памяти отводится под устройства ввода/вывода. Например, допустим, что адреса в диапазоне от 0x20000000 до 0x20FFFFFF используются для доступа к устройствам ввода/вывода. Каждому устройству ввода/вывода присваивается один или несколько адресов в этом диапазоне. При записи по заданному адресу данные отправляются в устройство. При чтении данные поступают из устройства. Этот метод связи с устройствами ввода/вывода называется вводом-выводом, отображенным в память (memory-mapped I/O, MMIO).



На рис. 9.1 изображены аппаратные средства, необходимые для поддержки двух устройств ввода/вывода, отображенных в память. Дешифратор адреса определяет, какое устройство нуждается в соединении с процессором. Он использует сигналы Address и MemWrite, чтобы генерировать управляющие сигналы для остальной части аппаратных средств. Мультиплексор ReadData выполняет переключение между памятью и различными устройствами ввода/вывода. Регистры с разрешением записи хранят значения, передаваемые в устройства ввода/вывода

Адреса, связанные с устройствами ввода/вывода, часто называют регистрами ввода/вывода, поскольку они на самом деле указывают на физические регистры в устройстве ввода/вывода, как показано на рис. 9.1.

Порты ввода/вывода общего назначения используются для чтения или записи цифровых сигналов. Как минимум выводам GPIO требуются отображенные в память регистры ввода/вывода для чтения логических уровней на входах порта, записи значений в выходы и установки направления вывода порта (вход или выход). Во многих встраиваемых системах отдельные выводы GPIO могут использоваться совместно с одним или несколькими периферийными устройствами специального назначения, поэтому необходимы дополнительные регистры конфигурации, от содержимого которых зависит, является ли вывод порта специализированным. Кроме того, процессор может прервать выполнение программы и обработать прерывание, когда на входном выводе возникает определенное событие, например нарастающий или спадающий фронт, а соответствующее условие прерывания указано в регистре конфигурации.

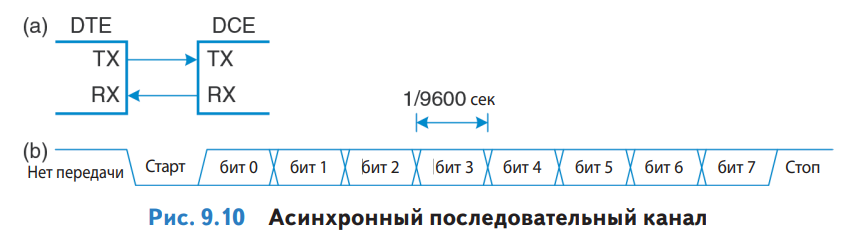
Доступ и изменение значений отображаемых в память регистров ввода/вывода осуществляются путем чтения или записи по соответствующим адресам памяти. На языке ассемблера это делается с помощью инструкций lw и sw. На языке C можно сделать то же самое при помощи указателей, но объявлять указатели для каждого отображаемого в память регистра ввода/вывода – утомительный и чреватый ошибками процесс. Более естественный способ описания и взаимодействия с отображаемыми в память регистрами ввода/вывода в языке C – использование структур.

Структуры в языке C – это способ сгруппировать коллекцию данных разных типов в единый блок. Использование структур в контексте отображаемых в память регистров позволяет обращаться к устройству ввода/вывода с использованием имени данного регистра или поля, а не адреса памяти. Программа на языке C может объявить структуру периферийного устройства, перечисляя регистры в том порядке, в котором они расположены в адресном пространстве памяти. Затем она может объявить указатель на такую структуру и получить доступ к периферийному устройству через указатель структуры.

Микроконтроллер может отправить несколько битов на внешнее устройство двумя способами – по нескольким проводам одновременно или посылая один бит за другим по одному проводу. Первый способ называется параллельным вводом-выводом, а второй – последовательным вводом-выводом. Последовательный ввод/вывод более популярен, особенно когда количество выводов ограничено, потому что он использует небольшое число проводов, но при этом его скорости достаточно для множества задач. На самом деле он настолько популярен, что для последовательного ввода/вывода было разработано множество стандартов, а микроконтроллеры содержат специальные аппаратные модули для простой передачи данных в соответствии с этими стандартами.

UART (произносится как «у-арт») – это периферийное устройство последовательного ввода/вывода для обмена данными между устройствами без использования тактового сигнала. Вместо этого системы должны заранее согласовать скорость передачи данных, и каждая из них должна локально генерировать свой собственный тактовый сигнал. Следовательно, такая передача является асинхронной, потому что тактовые сигналы не синхронизированы. Хотя системные тактовые сигналы могут иметь небольшую погрешность частоты и неизвестное соотношение фаз, UART обеспечивает надежную асинхронную связь. UART используется в таких протоколах, как RS-232 и RS-485.

Скорость передачи сигнала, измеряемая в бодах, – это количество передаваемых символов в секунду, тогда как скорость передачи данных измеряют в битах в секунду. В простой системе, такой как SPI, где каждый символ одновременно является битом данных, скорость передачи символов равна скорости передачи данных. В UART и некоторых других стандартах передачи сигналов в дополнение к данным передают служебные биты. Например, UART, который добавляет стартовые и стоповые биты для каждых 8 бит данных (т. е. передает 10 символов на 8 бит данных) и работает со скоростью 9600 бод, имеет скорость передачи (9600 символов/с) × (8 бит / 10 символов) = 7680 бит/с = 960 знаков/с.

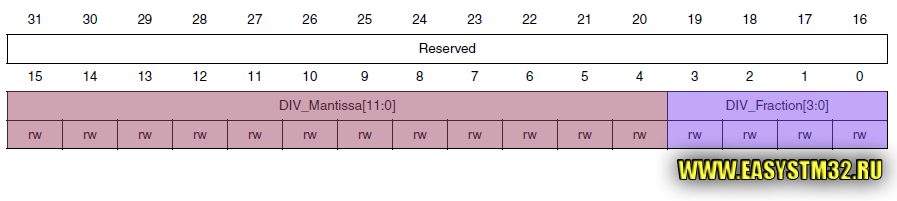


На рис. 9.10 (a) показан асинхронный последовательный канал. DTE передает данные в DCE по линии TX и получает данные обратно по линии RX. На рис. 9.10 (b) показана одна из этих линий при передаче данных со скоростью 9600 бод. В состоянии ожидания на линии присутствует постоянный уровень логической единицы. Каждый символ состоит из стартового бита (0), 7 или 8 бит данных, опционального бита четности и одного или нескольких стоповых битов (1). Чаще всего передают стартовые и стоповые биты и восемь бит данных. UART обнаруживает переход от состояния ожидания к началу передачи, чтобы своевременно начать процесс передачи данных. Хотя семи бит данных достаточно для отправки символа ASCII, обычно используются восемь бит, поскольку они позволяют передавать произвольный байт данных

И передатчик, и приемник должны быть настроены на одинаковую скорость передачи и количество битов данных, четность и стоповые биты, иначе данные будут искажены.

Стандарт RS-232 определяет несколько дополнительных сигналов. Сигналы запроса на передачу (request to send, RTS) и готовности к передаче (clear to send, CTS) могут использоваться для аппаратного подтверждения связи (handshaking). Они могут работать в одном из двух режимов. В режиме управления потоком DTE сбрасывает RTS в 0, когда он готов к приему данных от DCE. Точно так же DCE устанавливает CTS в 0, когда он готов к приему данных от DTE.

Начнем с регистра **USART\_BRR** при помощи которого задаётся скорость приёма/передачи:



Регистр делится на две части: Целая (**DIV\_Mantissa**) и дробная (**DIV\_Fraction**).

HAL\_UART\_Transmit() - функция блокирующая, то есть пока она не выполнится программа будет приостановлена.

Чтоб не блокировать программу на время отправки, нужно воспользоваться функцией с прерыванием по завершению передачи.

HAL\_UART\_Transmit\_IT(&huart1, (uint8\_t\*)"Hello World\n", 12);

По окончании отправки сработает прерывание и вызовет колбек:

void HAL\_UART\_TxCpltCallback(UART\_HandleTypeDef \*huart)  
{  
          if(huart == &huart1)  
          {  
                  // можно установить какой-то флаг, сообщающий об окончании отправки  
          }       
}

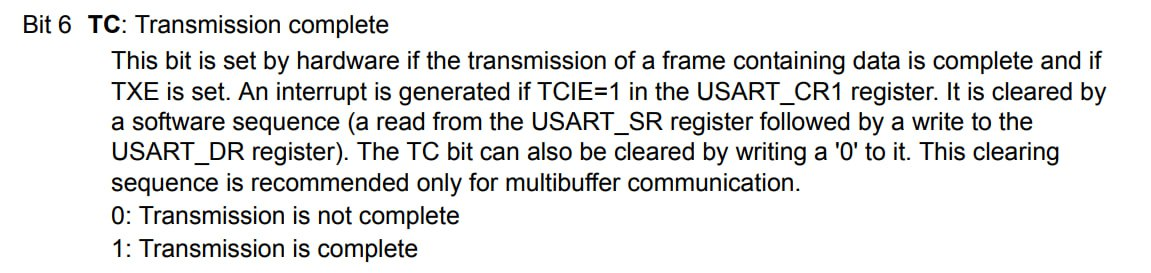
HAL\_UART\_Receive(&huart1, (uint8\_t\*)buff, 15, 1000);

*Второй аргумент — указатель на массив в который будут записываться полученные данные, третий — кол-во байт, которые нужно принять, последний — таймаут.*  
  
Эта функция тоже блокирующая, она будет ожидать до тех пор пока не получит все запрошенные байты, или пока не истечёт таймаут.

Если для отправки данных блокирующая функция вполне подходит, то для приёма совсем не годится *(мы ведь не знаем когда придут данные, если конечно они не идут сплошным потоком)*. Поэтому нужно использовать функцию вызывающую прерывание…

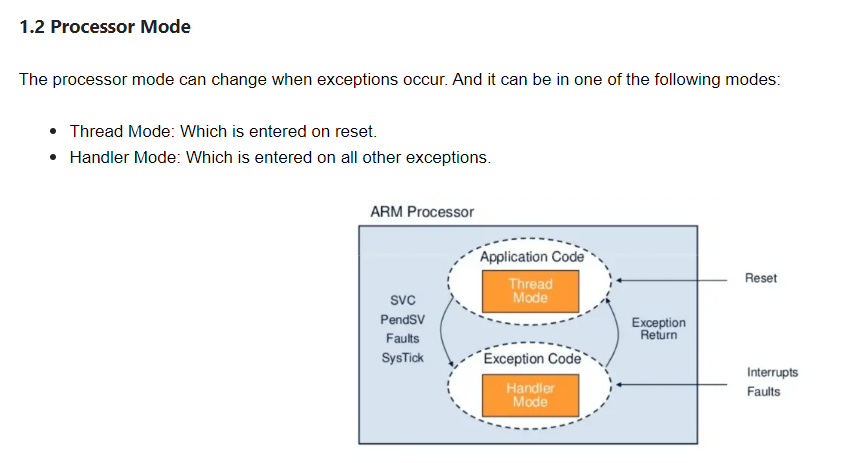
uint8\_t buff[16] = {0,};  
HAL\_UART\_Receive\_IT(&huart1, (uint8\_t\*)buff, 15);

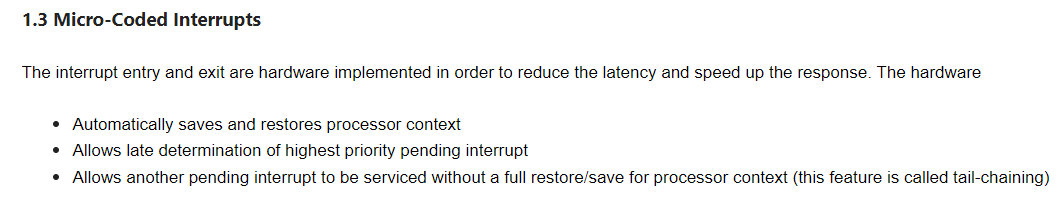
*Второй аргумент — указатель на массив в который будут записываться полученные данные, третий — кол-во байт, которые нужно принять.*  
  
Эта функция не блокирует программу. Прерывание произойдёт только после того, как всё запрошенные байты будут приняты.

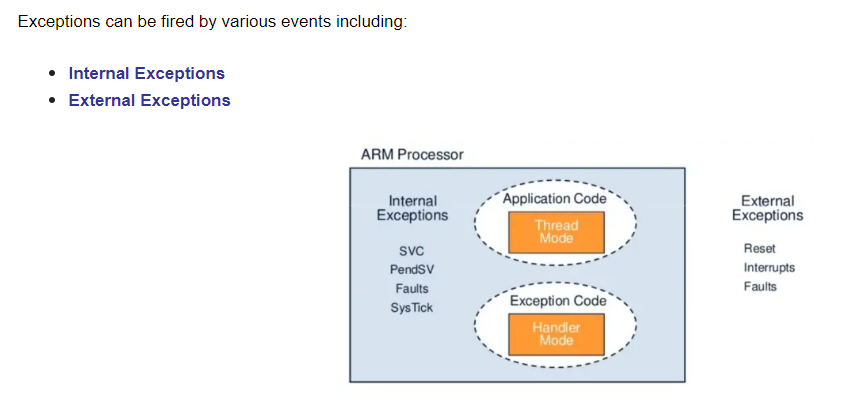


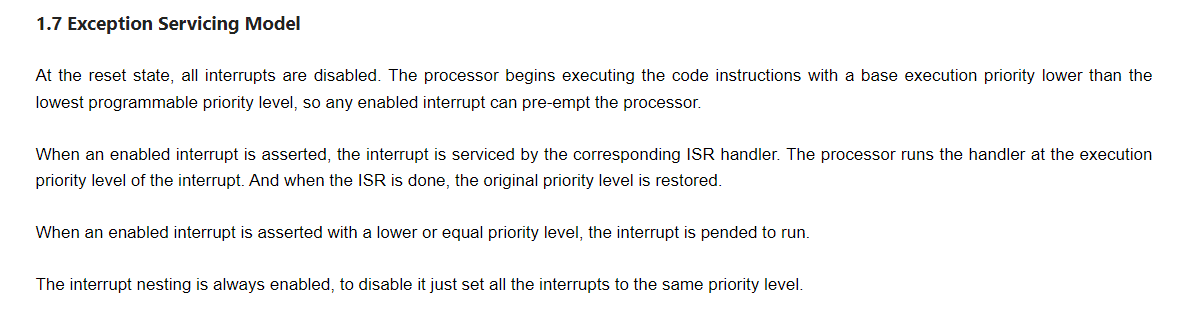
Исключение (exception) подобно незапланированному вызову функции, вызванному аппаратным или программным событием. Например, процессор может получить уведомление о том, что пользователь нажал клавишу на клавиатуре. В этом случае процессор может приостановить выполнение программы, определить нажатую клавишу и сохранить информацию об этом, после чего возобновить выполнение прерванной программы. Исключения, вызванные устройствами ввода-вывода, такими как клавиатура, часто называют прерываниями (interrupt).

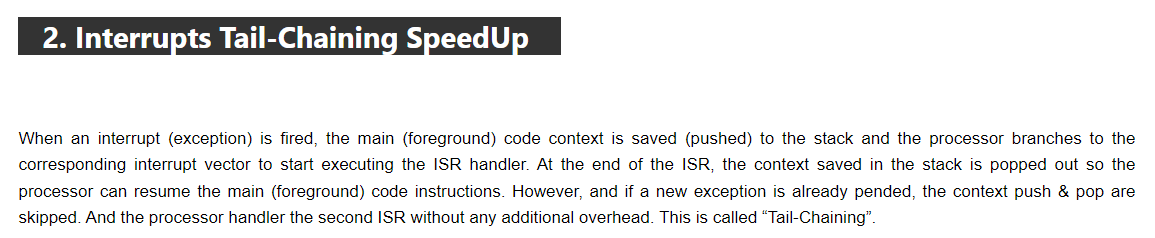
Прерывания – это аппаратные исключения

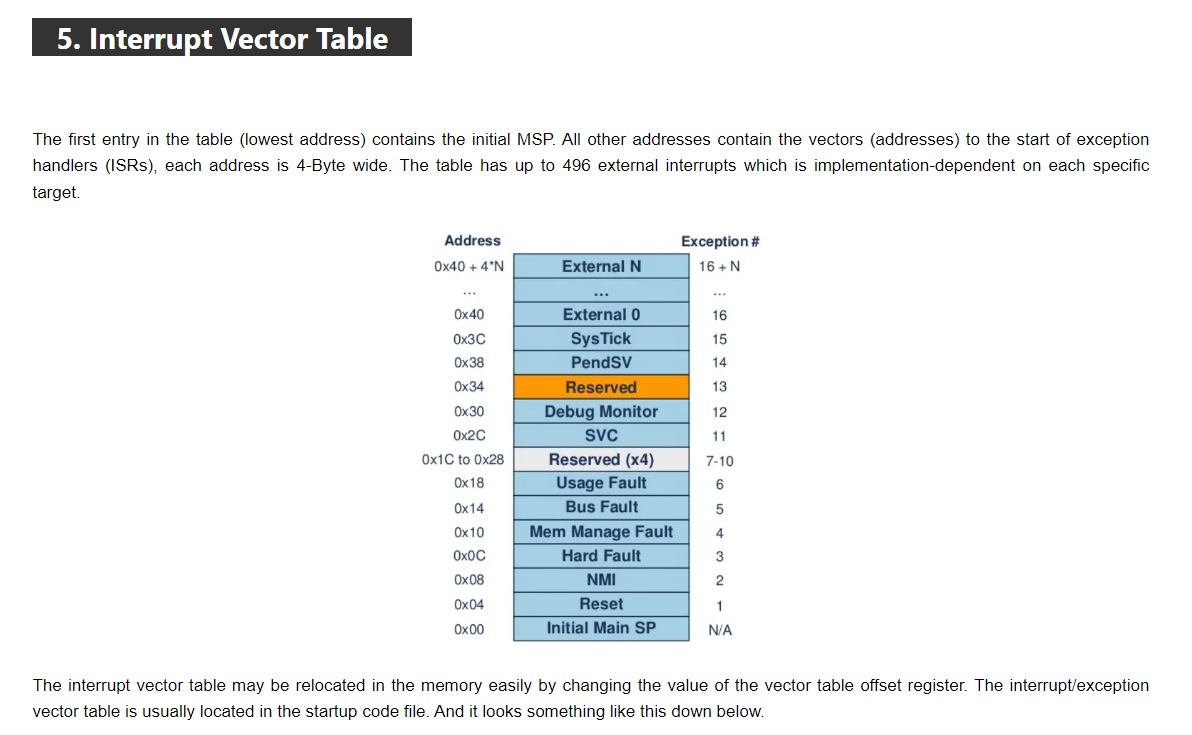
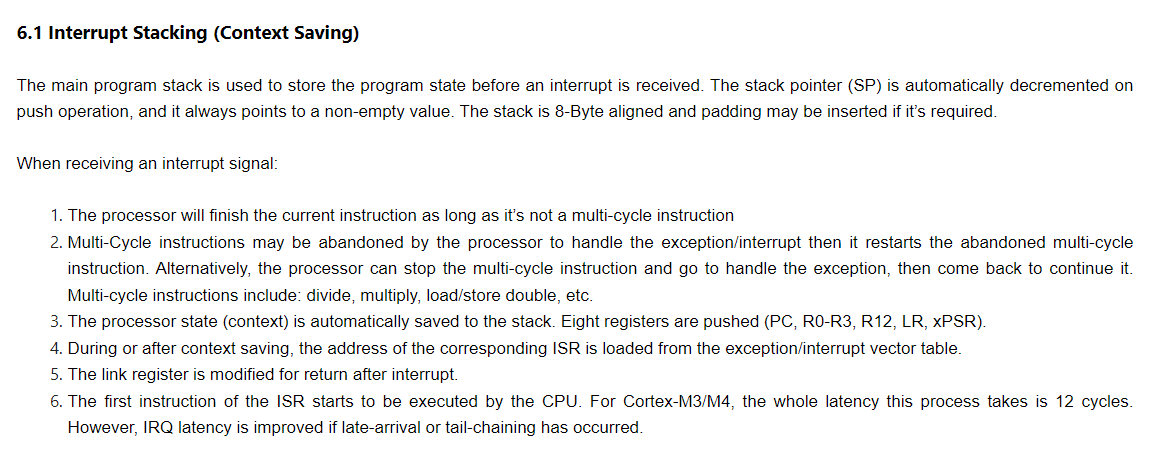


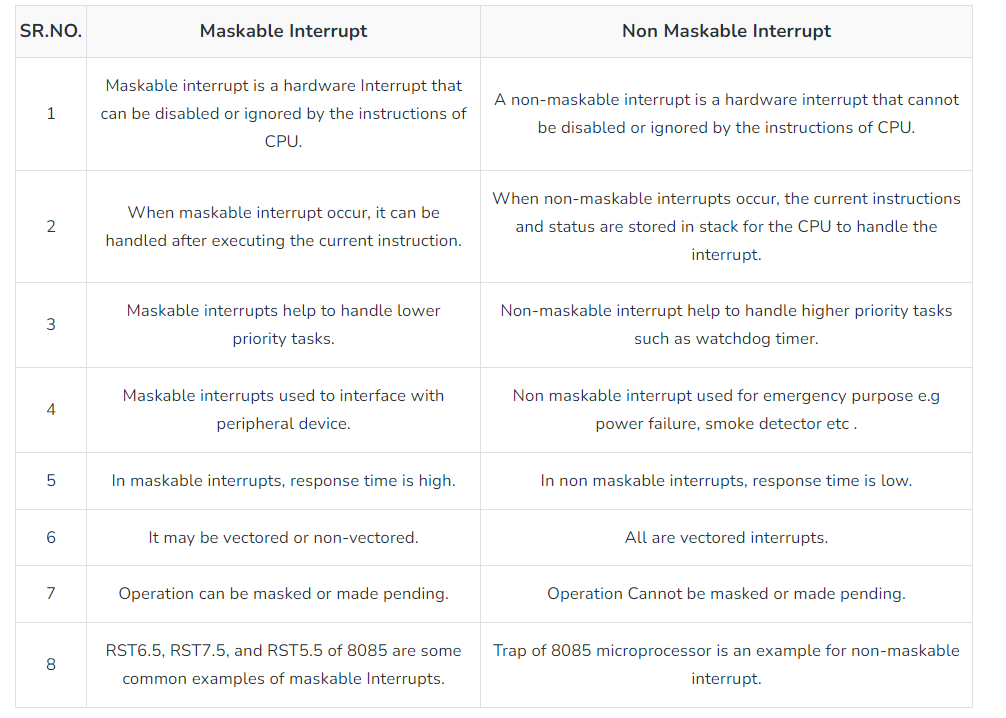
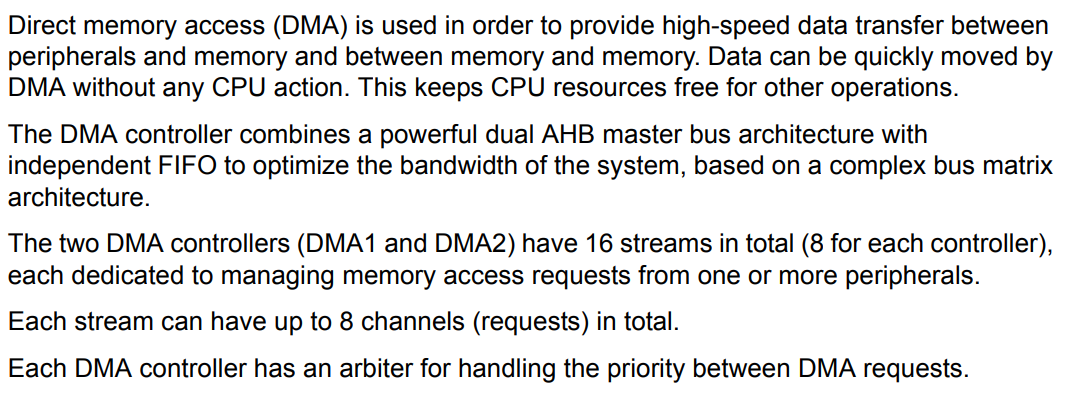


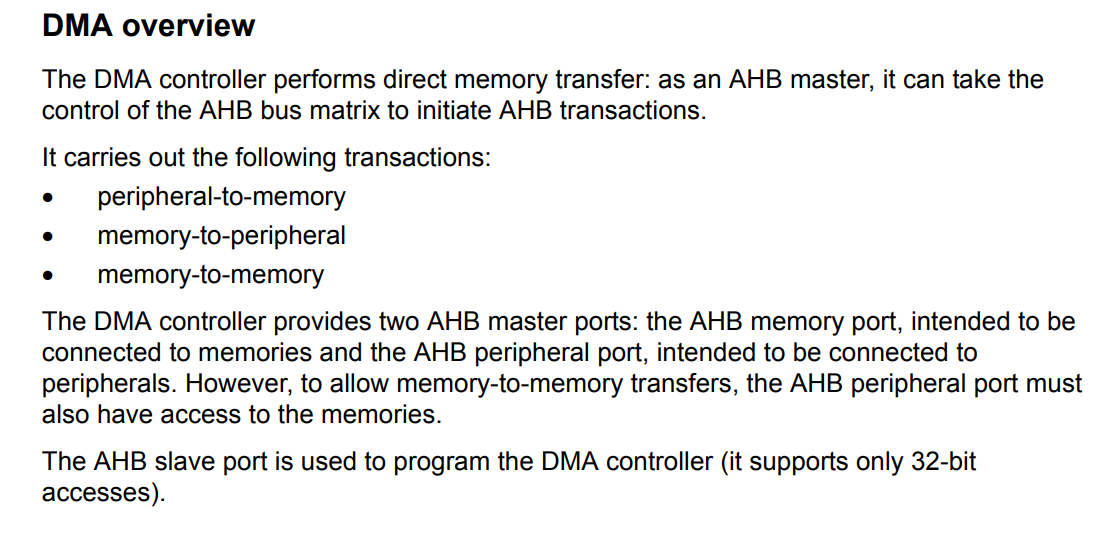


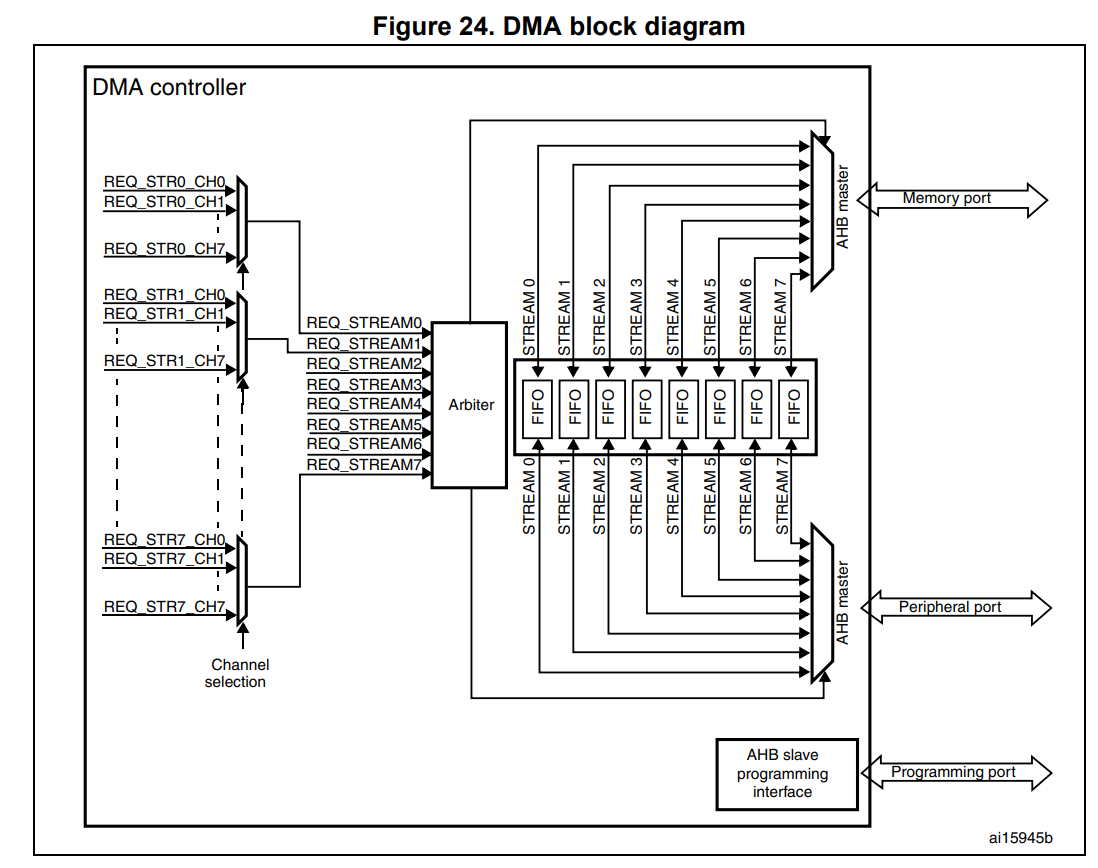


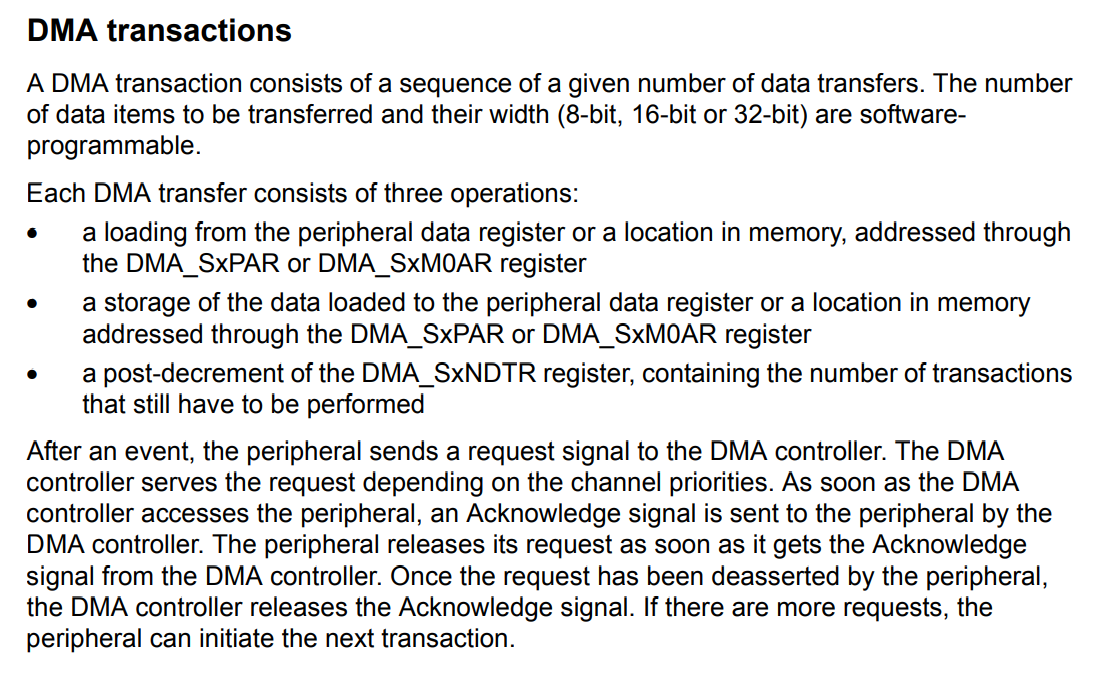


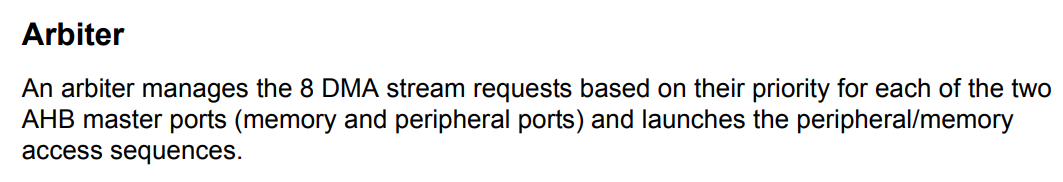
 

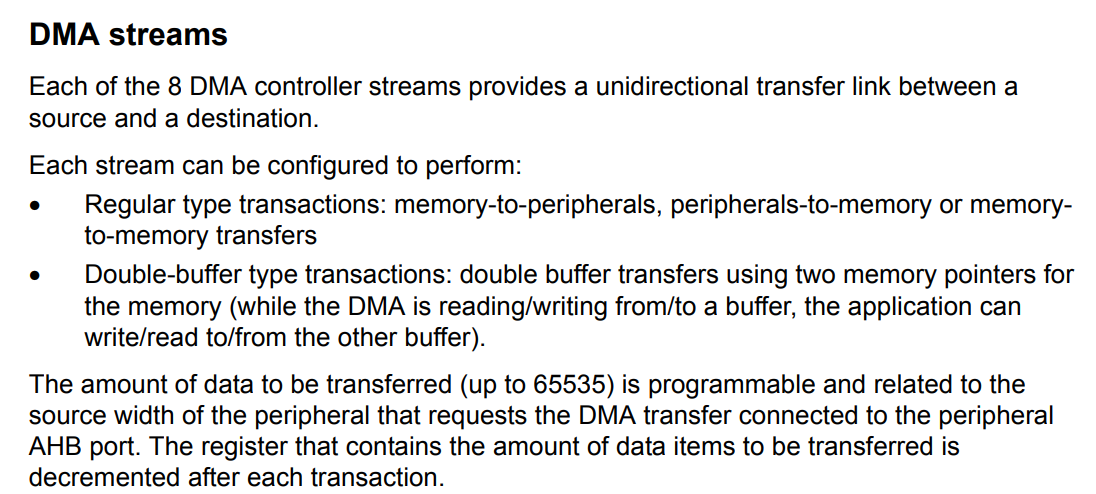
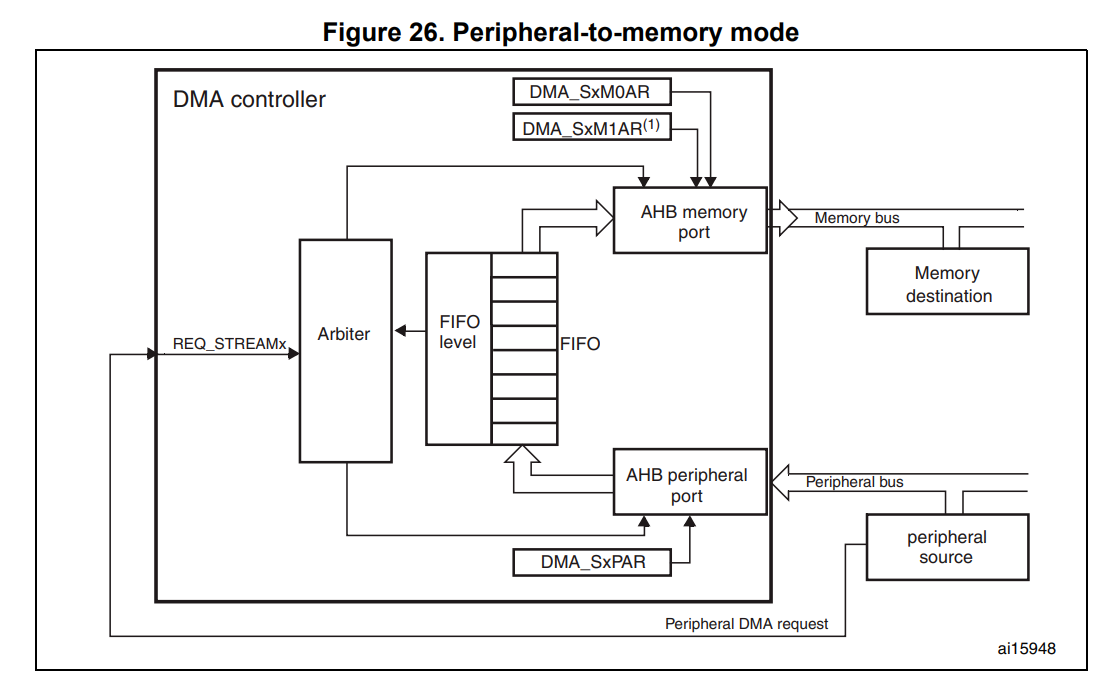
 

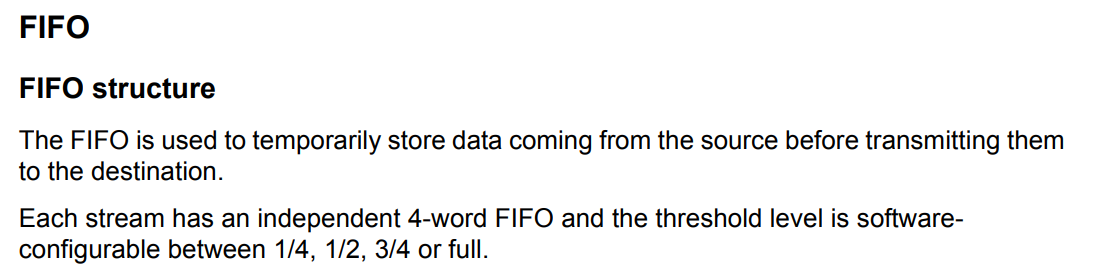


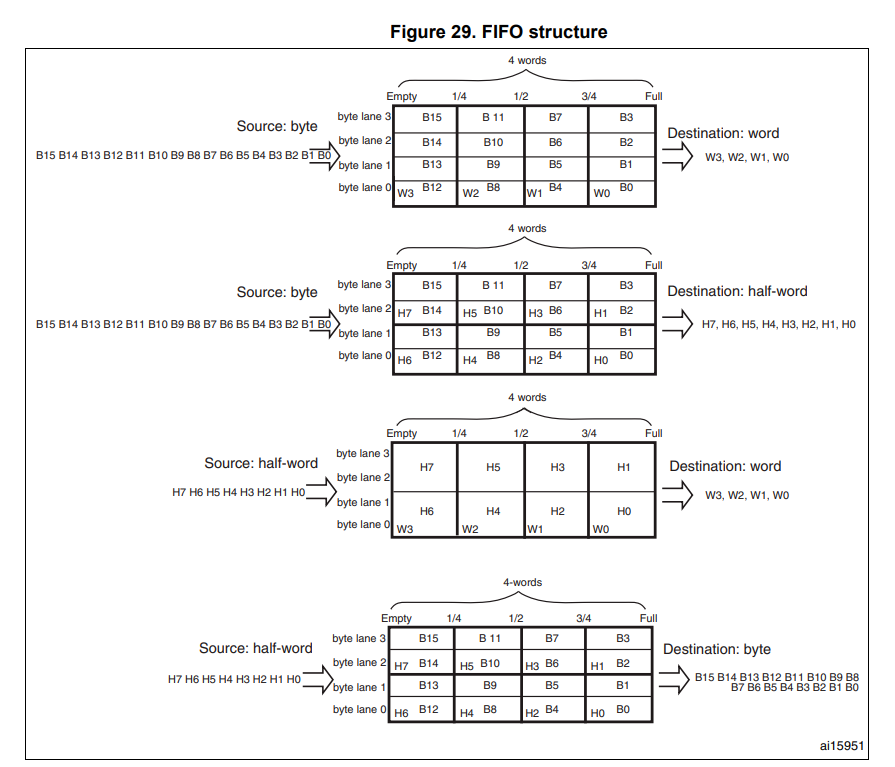


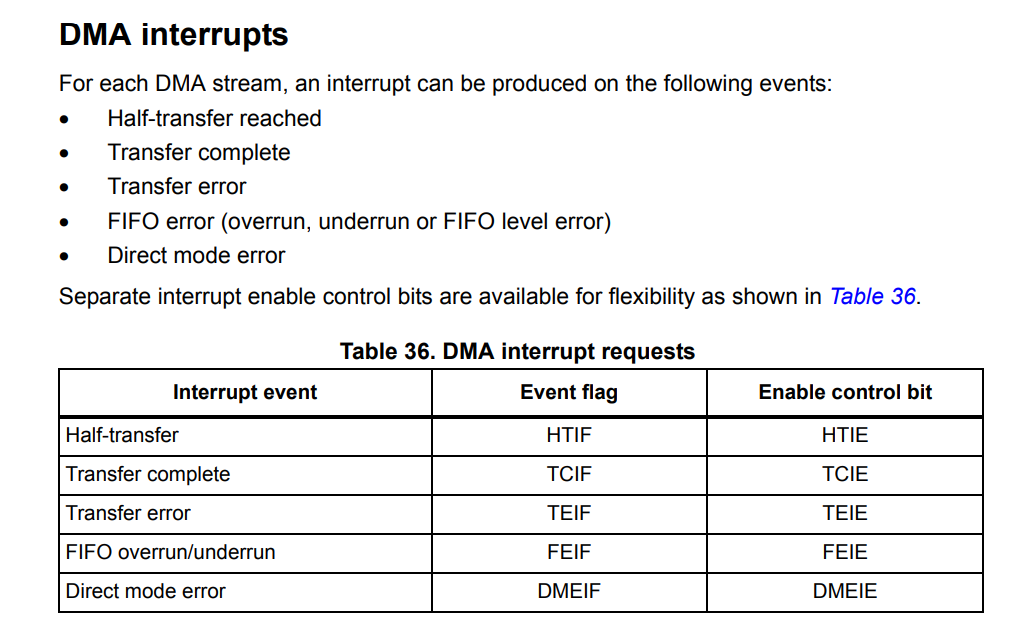










Встраиваемые системы обычно нуждаются в измерении времени. **Таймер** — это ~~адски крутая штуковина~~ просто счетчик, который при достижении заданного значения может вызывать определённые события — [прерывания](https://istarik.ru/blog/arduino/104.html)

**16-ти** битный таймер умеет считать в диапазоне от 0 до 65535 (это значение называется «переполнение» см. ниже). То есть, говоря простым языком, в пямяти есть переменная, которая увеличивается на единицу с каждым следующим «тиком» таймера. При достижении заданного пользователем значения (если значение не задавать, то счёт идет до максимального значения — 65535), после чего счётчик сбрасывается в ноль, генерируется какое-либо событие, и отсчёт начинается заново. Таймеры независимы друг от друга.  
Таймеры тактируются (то есть «тикают») от системной частоты и соответственно не потребляют ресурсов.

Таймеры stm32 делятся на три вида:  
  
Advanced-control timers (TIM1 и TIM8) — самые «нафаршированные».  
General-purpose timers (TIM2, TIM3, TIM4, TIM5) — функционал чуть меньше чем у предыдущих.  
Basic timers (все остальные) — сильно урезанный функционал.

