**Kendryte K210**

**SPI**

Для работы с любым модулем к210 необходим пакет sdk. Также можно подключить необходимые файлы для конкретного интерфейса. Для работы с SPI я подключила файлы из папки demo spi\_slave. В ней находятся файлы с функциями для работы как в режиме слейв, так и в режиме мастер. Режим задается дефайном в мейне. Также необходимо задать правильные номера ног для платы maix go в файле spi\_master.c.

Для работы с модулем RFID  нужно было написать свои функции для приема и передачи данных, т.к. стандартные не используют значение для регистра и задается 0 по умолчанию. Точно также нужно сделать для приема-передачи через дма. Эти функции написаны, но нужно написать свой вариант для задания регистра модуля.

С работой через прерывания возникли сложности, т.к. это плохо описано в документации и отсутствуют api функции для работы в режиме мастер.

Для работы прерываний необходимо вызвать функции, описанные в блоке PLIC документа Programming Guide, задать все необходимые значения и прописать функцию, которая будет вызываться из прерывания. В даташите говорится, что можно задать разные флаги для разных событий, но не указано, какие это флаги и как конкретно их нужно задать. Как мне кажется, маску нужно задать в регистре imr (SPI Interrupt Mask Register). Если записать там число 0х10 в функции инициализации spi\_init, то моя функция для прерывания вызывается после приема сообщения.

**I2C**

Поднять i2c, как он описан в Programming Guide в режиме мастер, инициализацией соответствующих ножек как функций i2c не получилось. Проверила пример из  Demo, где i2c работает инициализацией ног как gpio output, и далее вручную они поднимаются и опускаются по протоколу. Этот пример работает. Есть функции передачи и чтения просто значений, так и значений из заданного регистра слейв-устройства.

В режиме слейв нужно использовать те ножки, которые сконфигурированы для  i2c (через функцию fpioa) и функции из Demo i2c\_slave. Устройство должно работать следующим образом: При инициализации как слейв, задается прерывание и функция i2c\_slave\_irq, которая вызывается из прерывания. Эта функция читает статус регистр и, в зависимости от установленных флагов, должна вызывать соответствующие функции, но при попытках отправить сообщение и прочитать что-то мастером, устанавливается только флаг старт-сигнала. Последовательность от мастера приходит и устанавливаются биты ACK и NACK в конце последнего отправленного байта, как видно логическим анализатором.

I2C\_INTR\_STAT\_RX\_FULL

Насколько я поняла, функция приема должна вызываться при установке флага о заполнении буфера приема. Аргументом передается значение из регистра data\_cmd. Далее в этой функции проверяется переданное значение. Если это адрес регистра (возможно, номер слейв-устройства), то аргумент записывается как значение acces\_reg структуры \_slave\_info

{

    uint8\_t acces\_reg;

    uint8\_t reg\_data[SLAVE\_MAX\_ADDR];

} slave\_info\_t;

В противном случае, если адрес уже был получен, аргумент записывается в массив reg\_data и acces\_reg инкрементируется. Запись начинается с индекса массива, равному значению переданного аргумента в функцию и продолжается до индекса SLAVE\_MAX\_ADDR (равен 14).

Видимо, принятые значения потом можно прочитать из данной структуры.

I2C\_INTR\_STAT\_RD\_REQ

Когда установлен данный флаг, должна вызываться функция передачи i2c\_slave\_transmit(). Она читает данные из массива reg\_data и возвращает прочитанное значение, которое потом, в функции обработки прерывания записывается в регистр data\_cmd.

**UART**

Для работы с юартом на плате указаны ножки 4 и 5, и 6 и 7. Но в первом наборе не работает ножка 4 – RX.

Инициализируются ножки таким образом:

fpioa\_set\_function(6, FUNC\_UART1\_RX + UART\_NUM \* 2);

    fpioa\_set\_function(7, FUNC\_UART1\_TX + UART\_NUM \* 2);

инициализирующие функции:

uart\_init(UART\_NUM);

    uart\_configure(UART\_NUM, 115200, 8, UART\_STOP\_1, UART\_PARITY\_NONE);

Тестирование примера демо  **uart**. Отправка сообщения работает, прием – нет. Во время выполнения функции uart\_receive\_data не выполняется блок if.

int uart\_receive\_data(uart\_device\_number\_t channel, char \*buffer, size\_t buf\_len)

{

    size\_t i = 0;

    for(i = 0; i < buf\_len; i++)

    {

        if(uart[channel]->LSR & 1)

        {

                    buffer[i] = (char)(uart[channel]->RBR & 0xff);

        }

        else

        {

            break;

        }

    }

    return i;

}

С дма прием не работает также, отправка работает. Видимо, не установлен бит разрешения работы приема в соответствующем регистре. Для работы с дма необходимо вызвать функцию инициализации дма и вызвать функцию отправки с дма  соответствующими параметрами.

Тестирование примера  **uart\_interrupt**. Для работы через прерывания необходимо разрешить работу прерываний:

plic\_init();

    sysctl\_enable\_irq();

Далее вызывается функция задания триггера uart\_set\_send\_trigger(UART\_NUM, UART\_SEND\_FIFO\_0); есть 4 маски триггера, на как именно они отличаются друг от друга выяснить не удалось (не на длительность посылки). После этого вызываются функция uart\_irq\_register(UART\_NUM, UART\_SEND, on\_uart\_send, &v\_uart\_num, 2);, которая задает колбек после срабатывания триггера.

Также в этом файле протестировала работу функции uart\_send\_data\_dma\_irq. Для ее работы необходимо лишь вызвать функции инициализации прерываний и дма. В самой функции задается колбек после вызова прерывания.

Тестирование примера **uart\_dma\_irq**.  В этом примера вызывается функция uart\_handle\_data\_dma из файла uart.c, которая принимает в качестве аргументов структуры, с заданием необходимых параметров и функций: основной, например, функция приема или передачи данных, и функции, которая вызовется после окончания работы основной функции и вызова прерывания. В целом, похоже на работу из двух предыдущих примера, но теперь вся работа выполняется из одной функции.

Также имеется высокоскоростной юарт UARTHS. Для работы с ним необходимо подключить соответствующий хидер.