**official espnow example explanation**

* This example shows how to use ESPNOW.
* Prepare two device, one for sending ESPNOW data and another for receiving ESPNOW data.

**example\_espnow\_send\_cb()**

* creates an event, which contains useful information about the send event, the example\_espnow\_event\_t structure is defined by the creator of this program, it is defined in the header file
* creates a pointer with type example\_espnow\_event\_send\_cb\_t, the structure is defined in the header file, it points to a send\_cb type inside the info union inside the event type
* \*tx\_info is a pointer to a structure which contains information about the send event
* if its contents are NULL we got an error
* the event structure contains an id field, which identifies the event callback type (send or receive)
* copy the destination address from the tx\_info to our send\_cb structure to the mac\_addr field
* we also copy the status
* Post the event structure to a queue, which will be processed from a lower priority task
* if the queue is full, wait ESPNOW\_MAXDELAY ticks

**example\_espnow\_recv\_cb()**

* creates an event, which contains useful information about the receive event, the example\_espnow\_event\_t structure is defined by the creator of this program, it is defined in the header file
* creates a pointer with type example\_espnow\_event\_recv\_cb\_t, the structure is defined in the header file, it points to a recv\_cb type inside the info union inside the event type
* \*recv\_info is a pointer to a structure, which contains information about the receive event
* the structure contains the MAC address of the device, from whom the data is sent.
* the structure also contains the MAC address of the device to which the data is sent.
* If added a peer with encryption before, the receive packets may be encrypted as peer-to-peer message or unencrypted over the broadcast channel. Users can check the destination address to distinguish it.
* the event structure contains an id field, which identifies the event callback type (send or receive)
* copy the senders MAC address from the recv\_info structure to our defined recv\_cb structure
* allocate memory for the received data in the recv\_cb structure
* \*data points to the received data, we copy it to our recv\_cb structure
* also its length
* post the event to the queue to process it from a lower priority task
* if the queue is full, and it is not freed in ESPNOW\_MAXDELAY ticks, free the memory that was allocated for the data field in the recv\_cb structure

**example\_espnow\_data\_parse()**

* Parse received ESPNOW data. If we get a receive callback event from the queue, we use this function to process the data from the received packet
* example\_espnow\_data\_t is defined by the creator of this program, it is defined in the header file. \*data is a pointer to an uint8\_t type, we need to cast it to our predefined structure in order to process the data in the right way.
* too short data means error
* the following are pointers to variables defined outside of this function scope, these can be interpreted as out parameters of this function
* crc for data integrity
* save the crc from the data packet
* we need to set crc to 0 in order to recalculate crc properly.
* if data integrity is OK, return the type of the data packet (broadcast or unicast, (0,1))

**example\_espnow\_data\_prepare()**

* Prepare ESPNOW data to be sent. If we get a send callback event from the queue, we prepare the next packet, that has to be sent to the receiver
* example\_espnow\_send\_param\_t is a structure defined by the creator of this program, it is defined in the header file. It contains all the information (metadata) that has to be specified to send the data packet in a structured and safe way
* we load the data packet (buf) with the corresponding fields of the send\_param structure
* counter, which counts both unicast and broadcast data packets
* crc is calculated AFTER we loaded the packet with our payload
* Fill all remaining bytes after the data with random values for test purposes. example\_espnow\_data\_t has a size which only contains metadata about the packet, but send\_param structure contains the length
* calculate crc16

**example\_espnow\_task()**

* creates an event type
* out parameters from the parse function
* wait 5 seconds before starting
* the task is created with pv parameter, we cast it to send\_param\_t, it contains metadata about the packet which has to be sent
* send it to the MAC address specified in the send\_param
* before the task is created, the data is prepared with the helper function
* if there is an error, delete the task
* while there is event posted to the queue, do the following
* if the event is send callback type
* /create structure which reads the metadata about the callback event obtained from the queue
* flag, that indicates if the destination address is broadcast
* if destination address is broadcast, but the broadcast flag in the send parameter is false, break from the loop
* if the data sent is not broadcast (unicast or multicast), then decrement the packet counter in the send parameter structure
* if no packet left to be sent, delete the task, deinit the send parameter
* Delay a while before sending the next data.
* copy the destination MAC address from the send\_cb type to the send parameter structure, then prepare the send\_param structure, prepare the data
* Send the next data after the previous data is sent based on the prepared send\_param structure, if fail delete task and deinit send param structure
* if the event is receive callback type
* create structure which reads out metadata about the receive callback event obtained from the queue
* parse data from the received data, load the out type variables
* returns the type of the data (broadcast or unicast)
* free the allocated data, because we do not use it anymore, IMPORTANT
* If MAC address does not exist in peer list, add it to peer list.
* check if peer exists in the devices peer list
* allocate memory for peer info structure
* fill peer info structure with data about the peer
* free allocated memory for peer, because we saved it previously
* if the parse function returns with broadcast, process the data as broadcast
* if the magic number from the send\_param is bigger, than the magic number obtained from the data parse function
* copy the destination mac address obtained from the receive callback event to the send parameter structure, in order to prepare the next data to be sent. The receive callback event contains the source MAC address in the mac\_addr field, this way we can get from whom we got the data, this way we can start unicast communication with this peer (peer structure also contains this information)
* prepare the next data to be sent
* send the next data, if fails deinit send parameter and delete the task
* if we are already getting unicast, adjust the flags, and go on
* If receive unicast ESPNOW data, also stop sending broadcast ESPNOW data.

**app\_main()**

* WiFi should start before using ESPNOW
* pointer to a send param structure, this structure is defined by the creator of this program
* create a queue with ESPNOW\_QUEUE\_SIZE for the events