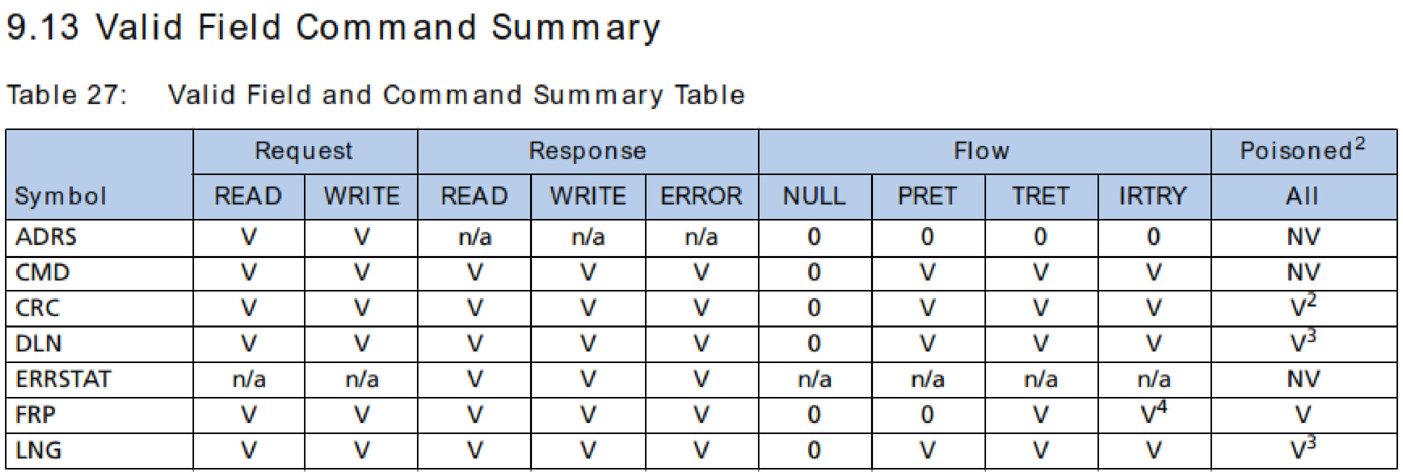
HMC Packet Item

Properties

Represents the unique fields of the packets for header and tail, with only stating a variable for a specific field once. In addition, a payload queue which is a multiple of 16-bytes or 128 bits FLITs



9.13 Valid Field Command Summary

Table 27: Valid Field and Command Summary Table

Symbol	Request		Response			Flow				Poisoned[2]
	READ	WRITE	READ	WRITE	ERROR	NULL	PRET	TRET	IRTRY	All
ADRS	V	V	n/a	n/a	n/a	0	0	0	0	NV
CMD	V	V	V	V	V	0	V	V	V	NV
CRC	V	V	V	V	V	0	V	V	V	V[2]
DLN	V	V	V	V	V	0	V	V	V	V[3]
ERRSTAT	n/a	n/a	V	V	V	n/a	n/a	n/a	n/a	NV
FRP	V	V	V	V	V	0	0	V	V[4]	V
LNG	V	V	V	V	V	0	V	V	V	V[3]

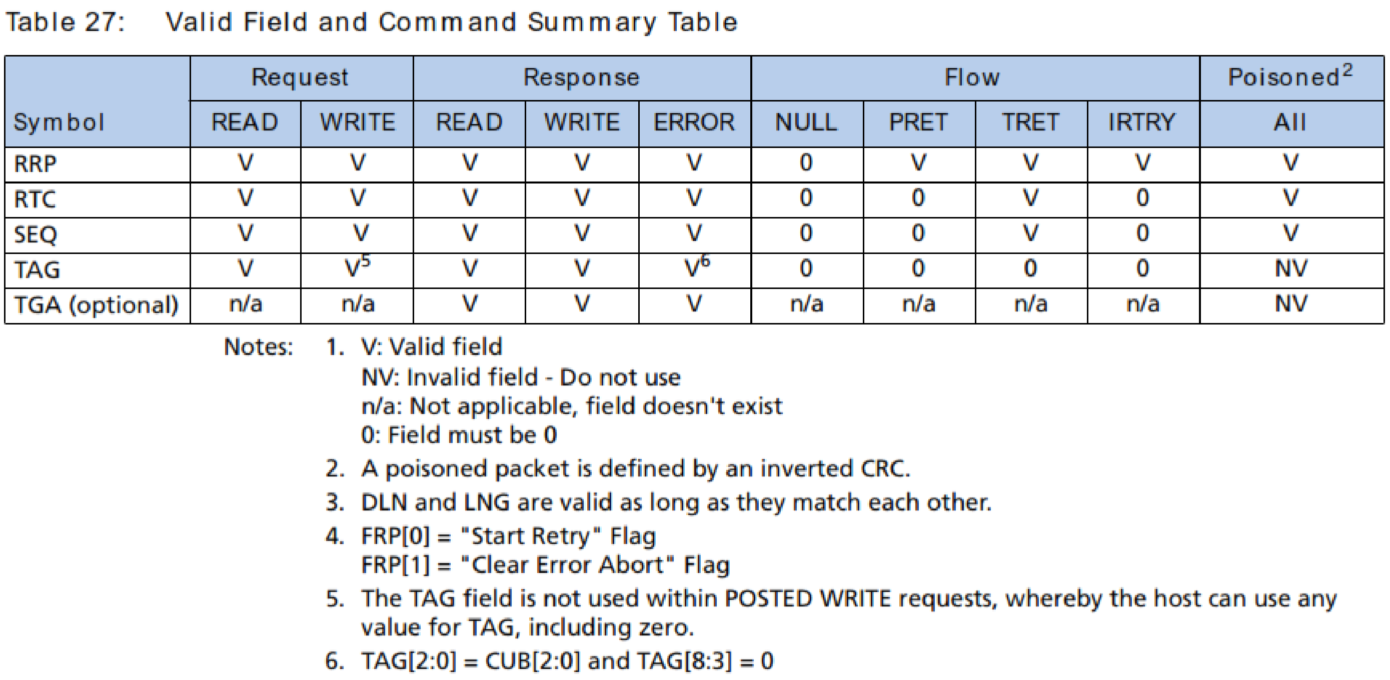
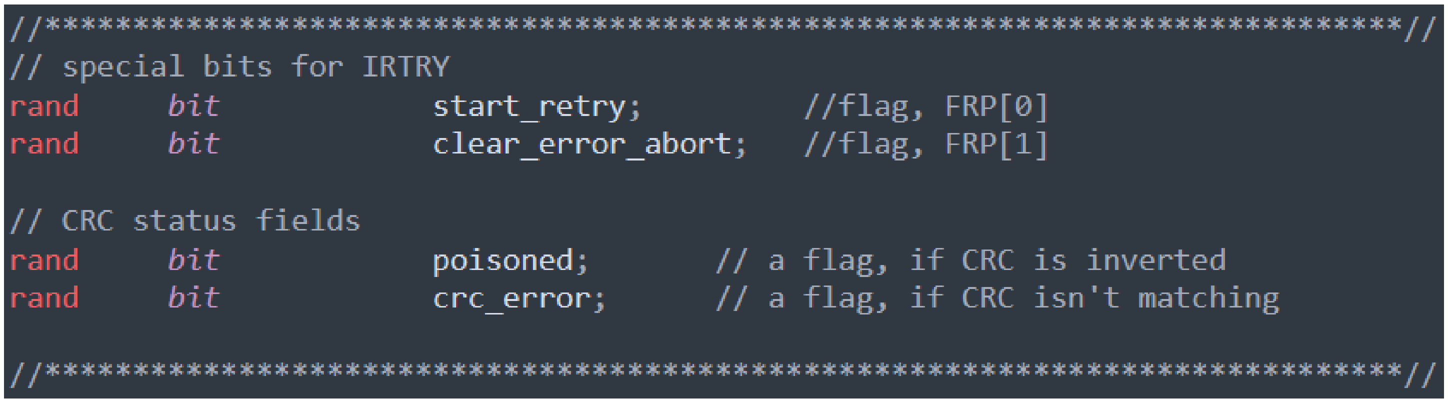


Table 27: Valid Field and Command Summary Table

Symbol	Request		Response			Flow				Poisoned[2]
	READ	WRITE	READ	WRITE	ERROR	NULL	PRET	TRET	IRTRY	All
RRP	V	V	V	V	V	0	V	V	V	V
RTC	V	V	V	V	V	0	0	V	0	V
SEQ	V	V	V	V	V	0	0	V	0	V
TAG	V	V[5]	V	V	V[6]	0	0	0	0	NV
TGA (optional)	n/a	n/a	V	V	V	n/a	n/a	n/a	n/a	NV

Notes:
1. V: Valid field
   NV: Invalid field - Do not use
   n/a: Not applicable, field doesn't exist
   0: Field must be 0
2. A poisoned packet is defined by an inverted CRC.
3. DLN and LNG are valid as long as they match each other.
4. FRP[0] = "Start Retry" Flag
   FRP[1] = "Clear Error Abort" Flag
5. The TAG field is not used within POSTED WRITE requests, whereby the host can use any value for TAG, including zero.
6. TAG[2:0] = CUB[2:0] and TAG[8:3] = 0

Special Properties



```
//***************************************************************************//
// special bits for IRTRY
rand     bit            start_retry;         //flag, FRP[0]
rand     bit            clear_error_abort;   //flag, FRP[1]

// CRC status fields
rand     bit            poisoned;       // a flag, if CRC is inverted
rand     bit            crc_error;      // a flag, if CRC isn't matching

//***************************************************************************//
```

For IRTRY flow packets, the FRP[7:0] flag in the tail will be zero except the FRP[0] and FRP[1] will be named after a special bits which are “start retry” and “clear error abort”.

A poisoned packet is identified by the fact its CRC is inverted. Receiving a poisoned packet doesn’t trigger a link retry

But if the CRC field is neither matches the correct CRC nor is inverted then the crc_error flag will be raised.

Constraints

Cube ID constraint

Since we are using only one agent to interact with the DUT so will set the ID to zero

Address constraint

For all flow commands the address is set to zero

A soft constraint about that for the actual memory inside the HMC device the don’t use the 4 LSB of the address so it is set to zero, but in our case, we don’t care much about the memory or mapping of the device connects to as it’s not related to DUT contents itself.

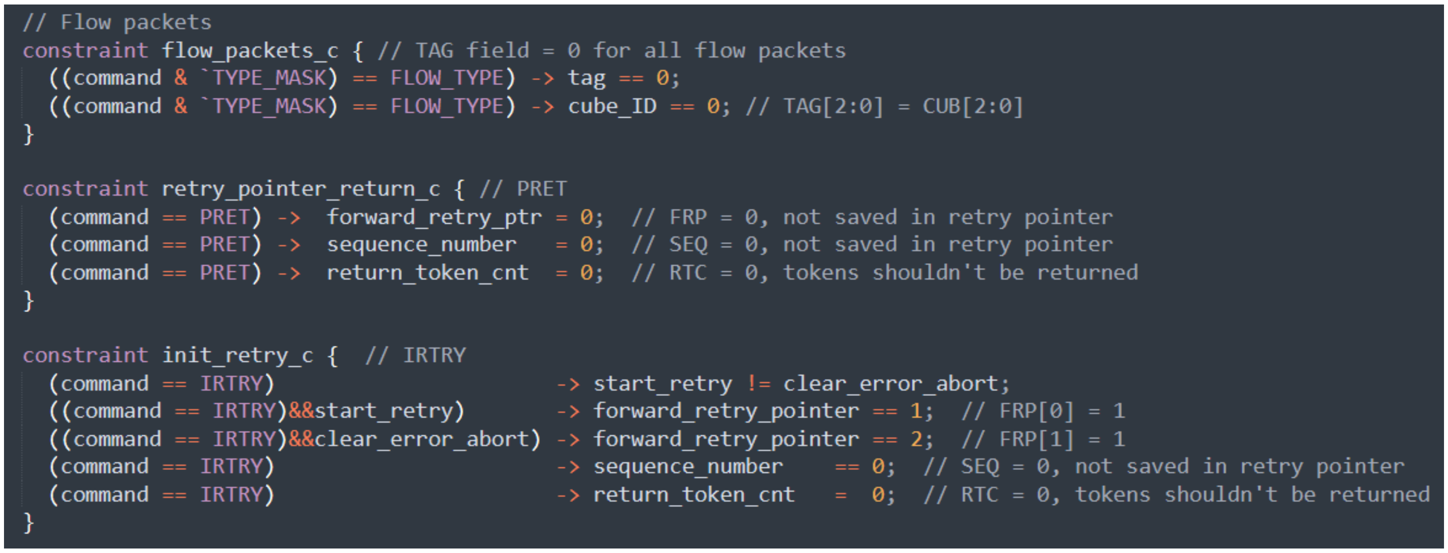
Duplicate Length constraint

DLN field must always be the same as LNG field.

Length constraint

For every packet type and command, the length field of the packet varies so must map all the commands and their corresponding length.

Constraints related to flow packets



```
// Flow packets
constraint flow_packets_c { // TAG field = 0 for all flow packets
  ((command & `TYPE_MASK) == FLOW_TYPE) -> tag == 0;
  ((command & `TYPE_MASK) == FLOW_TYPE) -> cube_ID == 0; // TAG[2:0] = CUB[2:0]
}

constraint retry_pointer_return_c { // PRET
  (command == PRET) ->  forward_retry_ptr = 0;  // FRP = 0, not saved in retry pointer
  (command == PRET) ->  sequence_number   = 0;  // SEQ = 0, not saved in retry pointer
  (command == PRET) ->  return_token_cnt  = 0;  // RTC = 0, tokens shouldn't be returned
}

constraint init_retry_c {  // IRTRY
  (command == IRTRY)                     -> start_retry != clear_error_abort;
  ((command == IRTRY)&&start_retry)       -> forward_retry_pointer == 1;  // FRP[0] = 1
  ((command == IRTRY)&&clear_error_abort) -> forward_retry_pointer == 2;  // FRP[1] = 1
  (command == IRTRY)                     -> sequence_number     == 0;  // SEQ = 0, not saved in retry pointer
  (command == IRTRY)                     -> return_token_cnt    =  0;  // RTC = 0, tokens shouldn't be returned
}
```

In flow packets, they set some fields to zero depending on the command, also for IRTRY command certain behavior is required depending on the settings of these fields and flags during randomization.

Methods

Post_randomize

After calling randomization, the length field is randomized and set, now we need to check for the size of the payload and if we need to fill it with FLITs at all

calculate_crc and calc_crc

responsible for calculating the crc they are based on the equation

$$x^{32} + x^{30} + x^{29} + x^{28} + x^{26} + x^{20} + x^{19} + x^{17} + x^{16} + x^{15} + x^{11} + x^{10} + x^{7} + x^{6} + x^{4} + x^{2} + x + 1$$

bit [32:0] polynomial = 33'b1_0111_0100_0001_1011_1000_1100_1101_0111

before doing pack to the item, after randomization, write

item.crc = item. calculate_crc();

it will assign a correct CRC to the CRC field inside the item after all other item fields are generated.

The following paragraph is taken as it is:

The CRC calculation operates on the LSB of the packet first. The packet CRC calculation must insert 0s in place of the 32-bits representing the CRC field before generating or checking the CRC. For example, when generating CRC for a packet, bits [63: 32] of the Tail presented to the

CRC generator should be all zeros. The output of the CRC generator will have a 32-bit CRC value that will then be inserted in bits [63:32] of the Tail before forwarding that FLIT of the packet.

When checking CRC for a packet, the CRC field should be removed from bits [63:32] of the Tail and replaced with 32-bits of zeros, then presented to the CRC checker. The output of the CRC checker will have a 32-bit CRC value that can be compared with the CRC value that was removed from the tail. If the two compare, the CRC check indicates no bit failures within the packet.

Do_pack

It packs the packet fields inside a uvm_packer called packer or just an empty array of bits.

Item.pack(Array);

Will automatically puts the header, data, tail fields in the correct order depending on the command type and save it inside the array.

Do_unpack

The opposite of pack, it takes an array of bits and extract from it the packet fields.

Item.unpack(Array);