Programmatic access to high speed hardware counters

RFC:

https://www.spinics.net/lists/linux-rdma/msg58579.html
To-date RDMA provides only counters at the whole port level

Verbs counters provide a way to count per-object information, with full HW offload

Observe behavior details of a single connection without requiring CPU involvement in each packet

Programmatic control allows process to manage counting as desired
**MOTIVATION #2**

**RDMA Debug-ability**
- Connect counters to objects in another process (Long term goal)
- Application self-debug details of the RDMA protocol hidden to the application (re-transmits, packet loss, NACKs, etc)

**Flow Processing**
- Passively monitor traffic flows, eg monitor networking on a per-VM basis
- DPDK

**Self-Monitoring**
- Compute actual instant bandwidth utilization
Counters objects hold a set of counter slots
Each slot can be assigned to a 'sample point'
API to read the counter value from all slots in a counter object
Basic counter object creation:

```c
struct ibv_counters *ibv_create_counters(struct ibv_context *context,
                                          struct ibv_counters_init_attr *init_attr);
int ibv_destroy_counters(struct ibv_counters *counters);
```
- Standard verbs sample points are intended to be very well defined.
- Easy to define hardware specific sampling points via a DV API.
- Starting out with simple packet and octet counters.
enum ibv_counter_description { 
    IBV_COUNTER_PACKETS, 
    IBV_COUNTER_BYTES, 
}

struct ibv_counter_attach_attr { 
    enum ibv_counter_description counter_desc; 
    uint32_t index; 
};

int ibv_attach_counters_point_flow(struct ibv_counters *counters, 
    struct ibv_counter_attach_attr *attr, 
    struct ibv_flow *flow);
- Expecting implementations to require a kernel syscall
- Return all counter values at once
- Approximate values or more expensive retrieval
- Simple monotonic and non-saturating uint64_t values
- HW not required to return an 'atomic snapshot'
API

Flags:

**IBV_READ_COUNTERS_ATTR_PREFER_CACHED**

```c
int ibv_read_counters(struct ibv_counters *counters, size_t ncounters,
                      uint64_t counters_value[], int attr_flags);
```
The API allows a wide range of combinations that hardware may not support:

- Combinations of sampling points in one object, eg cannot sample two flow objects at once
- Sampling types against objects, eg may support octet for flow but not for QP
- HW may not be able to attach/detach after object creation

App can detect this via the EOPNOTSUPP/EINVAL return code during setup.
FUTURE DIRECTIONS

- Monitor other IB objects, such as MR's CQs, SQs, etc.
- More standardized verbs counters