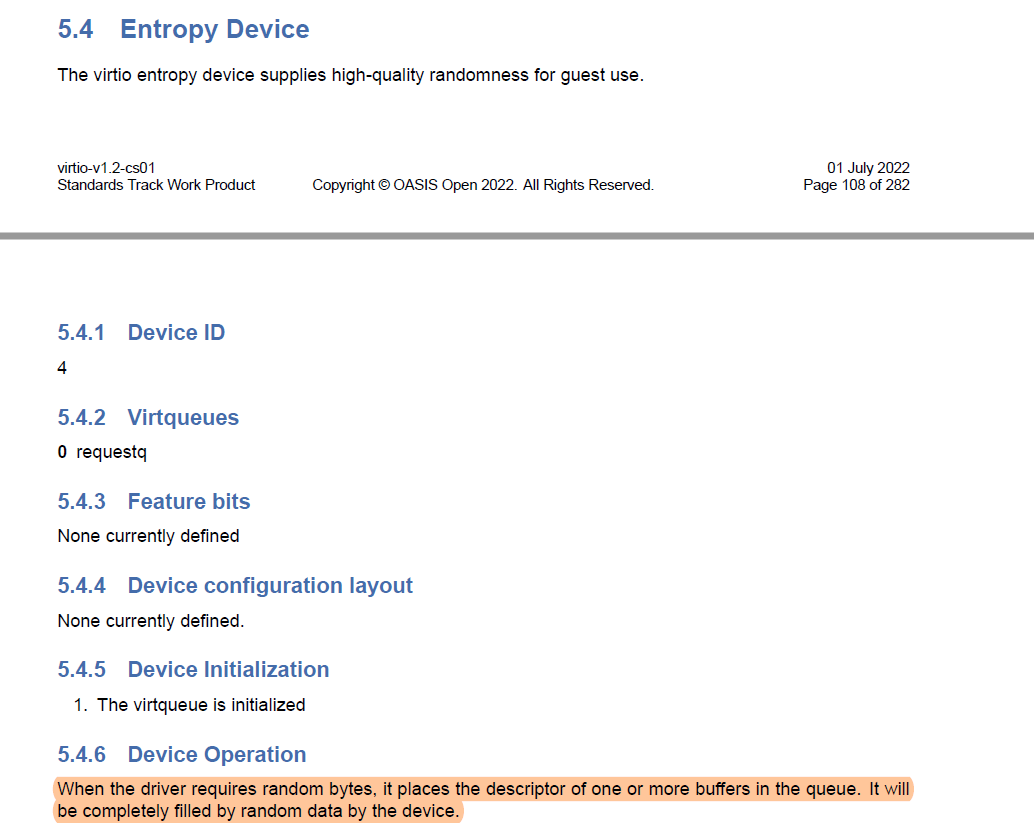
**VirtIO Rng**

**1 Spec**

编写Virtio Driver第一步是根据标准确认设备的ID，Virtqueue的个数和作用，操作的模式，比如RNG设备在标准中的描述如下图，可以知道RNG设备的ID是4，只有一个virtqueue，交互的方式是driver将buffer传递给device，device将随机数据填充到该buffer中后返回给driver



**2 初始化**

1. 驱动需要实现一个virtio\_register\_rng\_driver()函数，在virtio\_register\_drivers()函数中调用，如下：

|  |
| --- |
| C int virtio\_register\_drivers(void) {  int ret;  ... #ifdef CONFIG\_DRIVERS\_VIRTIO\_RNG  ret = virtio\_register\_rng\_driver();  if (ret < 0)  {  vrterr("virtio\_register\_rng\_driver failed, ret=%d\n", ret);  } #endif  ...  return ret; } |

1. 驱动在virtio\_register\_rng\_driver()中调用virtio\_register\_driver()将自己的驱动注册到virtio总线中去；
2. Virtio Device在初始化时，会将设备注册到virtio总线中去，触发virtio\_rng\_driver\_probe函数执行驱动的初始化，初始化完后会注册/dev/random到Vela的VFS中去；

**3 数据结构**

比较简单，只有设备名字

|  |
| --- |
| C struct virtio\_rng\_priv\_s {  FAR struct virtio\_device \*vdev;  char name[NAME\_MAX]; }; |

**4 Functions**

**virtio\_register\_rng\_driver()**

|  |
| --- |
| C static struct virtio\_driver g\_virtio\_rng\_driver = {  LIST\_INITIAL\_VALUE(g\_virtio\_rng\_driver.node), /\* node \*/  VIRTIO\_ID\_ENTROPY, /\* device id \*/  virtio\_rng\_probe, /\* probe \*/  virtio\_rng\_remove, /\* remove \*/ };  int virtio\_register\_rng\_driver(void) {  return virtio\_register\_driver(&g\_virtio\_rng\_driver); } |

**virtio\_rng\_driver\_probe()**

|  |
| --- |
| C static int virtio\_rng\_probe(FAR struct virtio\_device \*vdev) {  FAR struct virtio\_rng\_priv\_s \*priv;  FAR const char \*vqnames[1];  vq\_callback callback[1];  int ret;   priv = kmm\_zalloc(sizeof(\*priv));  if (priv == NULL)  {  vrterr("No enough memory\n");  return -ENOMEM;  }   priv->vdev = vdev;  vdev->priv = priv;   /\* Call openamp api to intialize the virtio deivce \*/   virtio\_set\_status(vdev, VIRTIO\_CONFIG\_STATUS\_DRIVER);  virtio\_set\_features(vdev, 0);  virtio\_set\_status(vdev, VIRTIO\_CONFIG\_FEATURES\_OK);   vqnames[0] = "virtio\_rng\_rx";  callback[0] = virtio\_rng\_done;  ret = virtio\_create\_virtqueues(vdev, 0, 1, vqnames, callback);  if (ret < 0)  {  vrterr("virtio\_device\_create\_virtqueue failed, ret=%d\n", ret);  goto err\_with\_priv;  }   virtio\_set\_status(vdev, VIRTIO\_CONFIG\_STATUS\_DRIVER\_OK);  virtqueue\_enable\_cb(vdev->vrings\_info[0].vq);   /\* Register NuttX driver \*/   if (g\_virtio\_rng\_idx == 0)  {  strlcpy(priv->name, "/dev/random", NAME\_MAX);  }  else  {  snprintf(priv->name, NAME\_MAX, "/dev/random%d", g\_virtio\_rng\_idx);  }   ret = register\_driver(priv->name, &g\_virtio\_rng\_ops, 0444, priv);  if (ret < 0)  {  vrterr("Register NuttX driver failed, ret=%d\n", ret);  goto err\_with\_virtqueue;  }   g\_virtio\_rng\_idx++;  return ret;  err\_with\_virtqueue:  virtio\_reset\_device(vdev);  virtio\_delete\_virtqueues(vdev); err\_with\_priv:  kmm\_free(priv);  return ret; } |

**g\_rng\_ops**

rng只需要实现read的operation即可，如下

|  |
| --- |
| C static const struct file\_operations g\_virtio\_rng\_ops = {  NULL, /\* open \*/  NULL, /\* close \*/  virtio\_rng\_read, /\* read \*/  NULL, /\* write \*/  NULL, /\* seek \*/  NULL, /\* ioctl \*/  NULL, /\* mmap \*/  NULL, /\* truncate \*/  NULL, /\* poll \*/ #ifndef CONFIG\_DISABLE\_PSEUDOFS\_OPERATIONS  NULL, /\* unlink \*/ #endif }; |

**static ssize\_t virtio\_rng\_read(FAR struct file \*filep, FAR char \*buffer, size\_t buflen)**

描述：将user的buffer传递给device端，等待device端填充随机数，device端返回后，buffer被填充为随机数，之后返回给user

|  |
| --- |
| C struct virtio\_rng\_cookie\_s {  sem\_t sem;  size\_t len; };  static ssize\_t virtio\_rng\_read(FAR struct file \*filep, FAR char \*buffer,  size\_t buflen) {  FAR struct virtio\_rng\_priv\_s \*priv = filep->f\_inode->i\_private;  FAR struct virtqueue \*vq = priv->vdev->vrings\_info[0].vq;  struct virtio\_rng\_cookie\_s cookie;  struct virtqueue\_buf vb;  int ret;   /\* Init the cookie \*/   cookie.len = 0;  nxsem\_init(&cookie.sem, 0, 0);   /\* Add the input buffer to the virtqueue, and the cookie as the virtqueue  \* cookie. (virtqueue\_get\_buffer() will return cookie).  \*/   vb.buf = buffer;  vb.len = buflen;  ret = **virtqueue\_add\_buffer(vq, &vb, 0, 1, &cookie);**  if (ret < 0)  {  return ret;  }   /\* Notify the other side to process the added virtqueue buffer \*/   **virtqueue\_kick(vq);**   /\* Wait fot completion \*/   nxsem\_wait\_uninterruptible(&cookie.sem);  return cookie.len; } |

**static void virtio\_rng\_done(FAR struct virtqueue \*vq)**

描述：在device端填充完成后，通知driver端，触发该函数

|  |
| --- |
| C static void virtio\_rng\_done(FAR struct virtqueue \*vq) {  FAR struct virtio\_rng\_cookie\_s \*cookie;  uint32\_t len;   /\* Get the buffer, virtqueue\_get\_buffer() return the cookie added in  \* virtio\_rng\_read().  \*/   cookie = virtqueue\_get\_buffer(vq, &len, NULL);  if (cookie != NULL)  {  /\* Assign the return length \*/   cookie->len = len;   /\* Read completed, post the sem \*/   nxsem\_post(&cookie->sem);  } } |

QEMU上的配置和使用参考[Rng](https://xiaomi.f.mioffice.cn/wiki/wikk4icnzDocTXLwD1nM6w8GIhh)，驱动设计参考[RNG](https://xiaomi.f.mioffice.cn/wiki/wikk4HzZp7gQCxwe5eMhl0Znu4b) 。