**VirtIO GPU**

1. **背景概要**

在Vela上实现了基于[VirtIO 1.2标准](https://docs.oasis-open.org/virtio/virtio/v1.2/cs01/virtio-v1.2-cs01.html#x1-3650007)的通过MMIO通信的VirtIO-GPU驱动，实现将Vela上的framebuffer内容搬移到后端（QEMU）显示的功能。

2. **设计方案**

[VirtIO-GPU及图形框架设计](https://xiaomi.f.mioffice.cn/wiki/wikk4I13GVFCawJ50rFX6EBOLkc) [Display驱动](https://xiaomi.f.mioffice.cn/wiki/wikk4z5NC8bLv0ITzQFoXkDSYnc)

3. **实现细节**

3.1 **工作流程**

同步（阻塞）操作 / 异步（非阻塞）操作

3.1.1 **初始化**

1. virtio\_gpu\_init - 初始化VirtIO队列及回调
2. virtio\_gpu\_get\_display\_info - 获取显示器信息
3. virtio\_gpu\_create\_2d - 在host上创建一个resource，获得资源id
4. virtio\_gpu\_attach\_backing - 给刚创建的resource连接guest(Vela)分配的(frame)buffer作为backing
5. virtio\_gpu\_set\_scanout - 将resource与scanout（host端的输出）对接起来
6. virtio\_gpu\_transfer\_to\_host\_2d - 将backing buffer的内容传输到host（初始帧）
7. virtio\_gpu\_flush\_resource - 通知host进行resource到scanout的显示刷新
8. fb\_register - 注册Vela内部的framebuffer设备

3.1.2 **刷新帧**

1. virtio\_gpu\_transfer\_to\_host\_2d - 将framebuffer的内容传输到host
2. virtio\_gpu\_flush\_resource - 通知host进行显示刷新

3.2 **数据结构**

|  |
| --- |
| C struct virtio\_gpu\_priv\_s {  struct fb\_vtable\_s vtable; /\* Must be cast compatible with virtio\_gpu\_priv\_s \*/  FAR struct virtio\_device \*vdev; /\* Contained virtio device \*/  FAR uint8\_t \*fbmem; /\* Allocated framebuffer \*/  size\_t fblen; /\* Size of the framebuffer in bytes \*/  fb\_coord\_t xres; /\* Horizontal resolution in pixel columns \*/  fb\_coord\_t yres; /\* Vertical resolution in pixel rows \*/  fb\_coord\_t stride; /\* Width of a row in bytes \*/  uint8\_t display; /\* Display number \*/ }; |

struct virtio\_gpu\_priv\_s为VirtIO-GPU驱动的基本数据存储，包括初始化的struct virtio\_device和framebuffer信息。其中struct fb\_vtable\_s需要放在第一位，以便于在执行framebuffer相关操作时通过指针映射拿到struct virtio\_gpu\_priv\_s的信息。

|  |
| --- |
| C struct virtio\_gpu\_cookie\_s {  bool blocking;  FAR void \*p; }; |

struct virtio\_gpu\_cookie\_s为驱动发送命令给VirtIO总线时传递cookie信息的结构体。

* blocking == true时，表明这是一个同步操作，此时指针p指向同步信号量；
* blocking == false时，表明这是一个异步操作，此时指针p指向需要释放的缓冲区。

|  |
| --- |
| C struct virtio\_gpu\_backing\_s {  struct virtio\_gpu\_resource\_attach\_backing cmd;  struct virtio\_gpu\_mem\_entry ents[VIRTIO\_GPU\_MAX\_NENTS]; }; |

struct virtio\_gpu\_backing\_s为[**VIRTIO\_GPU\_CMD\_RESOURCE\_ATTACH\_BACKING**](https://docs.oasis-open.org/virtio/virtio/v1.2/cs01/virtio-v1.2-cs01.html#x1-3810008)操作实现需要的结构体。按照VirtIO规定，传输的结构体为命令header加其后紧随的数个struct virtio\_gpu\_mem\_entry。

3.3 **函数操作**

3.3.1 **发送命令与回调**

基础命令操作函数大致如下：

|  |
| --- |
| C static int virtio\_gpu\_xxxx(FAR struct virtio\_gpu\_priv\_s \*priv, ...) {  FAR struct virtqueue \*vq = priv->vdev->vrings\_info[VIRTIO\_GPU\_CTL].vq;  struct virtio\_gpu\_xxxx cmd;  struct virtio\_gpu\_ctrl\_hdr resp;  struct virtqueue\_buf vb[2];  int ret;   memset(&cmd, 0, sizeof(cmd));  cmd.hdr.type = VIRTIO\_GPU\_CMD\_XXXX;  cmd.xxx = yyy;  ...    vb[0].buf = &cmd;  vb[0].len = sizeof(cmd);  vb[1].buf = &resp;  vb[1].len = sizeof(resp);   ret = virtio\_gpu\_send\_cmd(vq, vb, 1, 1, NULL);  if (ret >= 0 && resp.type != VIRTIO\_GPU\_RESP\_OK\_NODATA)  {  ret = VIRTIO\_GPU\_MAP\_ERR(resp.type);  }   return ret; } |

以上为一个常规的同步命令函数，即需要等待命令完成才返回，由virtio\_gpu\_send\_cmd的第四个参数决定，该参数标记了需要host写入的buffer数量，大于0时表示我们驱动侧提供了至少一个buffer供host写入，也即我们需要返回结果，所以判定为同步操作。如需改为异步（非阻塞）操作，则只需进行如下修改：

|  |
| --- |
| C  struct virtqueue\_buf vb; ...  vb.buf = &cmd;  vb.len = sizeof(cmd);  ret = virtio\_gpu\_send\_cmd(vq, vb, 1, 0, NULL); |

virtio\_gpu\_send\_cmd具体实现如下：

|  |
| --- |
| C  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Name: virtio\_gpu\_send\_cmd  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  static int virtio\_gpu\_send\_cmd(FAR struct virtqueue \*vq,  FAR struct virtqueue\_buf \*buf\_list,  int readable, int writable, FAR void \*buf) {  int ret;   if (writable > 0)  {  sem\_t sem;  struct virtio\_gpu\_cookie\_s cookie;   sem\_init(&sem, 0, 0);  cookie.blocking = true;  cookie.p = &sem;  ret = virtqueue\_add\_buffer(vq, buf\_list, readable, writable, &cookie);  if (ret >= 0)  {  virtqueue\_kick(vq);  sem\_wait(&sem);  }   sem\_destroy(&sem);  }  else  {  FAR struct virtio\_gpu\_cookie\_s \*cookie;   cookie = kmm\_malloc(sizeof(\*cookie));  if (cookie == NULL)  {  vrterr("ERROR: Failed to allocate cookie memory");  ret = -ENOMEM;  }  else  {  cookie->blocking = false;  cookie->p = buf;  ret = virtqueue\_add\_buffer(vq, buf\_list, readable, writable,  cookie);  if (ret >= 0)  {  virtqueue\_kick(vq);  }  else  {  virtio\_free\_buf(vq->vq\_dev, buf);  kmm\_free(cookie);  }  }  }   return ret; } |

Control Queue回调函数：

|  |
| --- |
| C /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Name: virtio\_gpu\_done  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  static void virtio\_gpu\_done(FAR struct virtqueue \*vq) {  FAR struct virtio\_gpu\_cookie\_s \*cookie;   while ((cookie = virtqueue\_get\_buffer(vq, NULL, NULL)) != NULL)  {  if (cookie->blocking)  {  sem\_post((FAR sem\_t \*)cookie->p);  }  else  {  virtio\_free\_buf(vq->vq\_dev, cookie->p);  kmm\_free(cookie);  }  } } |

3.3.2 **virtio\_gpu\_init - 初始化VirtIO队列及回调**

|  |
| --- |
| C  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Name: virtio\_gpu\_init  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  static int virtio\_gpu\_init(FAR struct virtio\_gpu\_priv\_s \*priv,  FAR struct virtio\_device \*vdev) {  FAR const char \*vqnames[VIRTIO\_GPU\_NUM];  vq\_callback callbacks[VIRTIO\_GPU\_NUM];  int ret;   priv->vdev = vdev;  vdev->priv = priv;   /\* Initialize the virtio device \*/   virtio\_set\_status(vdev, VIRTIO\_CONFIG\_STATUS\_DRIVER);  virtio\_set\_features(vdev, 0);  virtio\_set\_status(vdev, VIRTIO\_CONFIG\_FEATURES\_OK);   vqnames[VIRTIO\_GPU\_CTL] = "virtio\_gpu\_ctl";  callbacks[VIRTIO\_GPU\_CTL] = virtio\_gpu\_done;  ret = virtio\_create\_virtqueues(vdev, 0, VIRTIO\_GPU\_NUM, vqnames,  callbacks);  if (ret < 0)  {  vrterr("virtio\_device\_create\_virtqueue failed, ret=%d", ret);  return ret;  }   virtio\_set\_status(vdev, VIRTIO\_CONFIG\_STATUS\_DRIVER\_OK);  return OK; } |

目前仅实现了Control Queue的2D部分功能，未来添加Cursor Queue及3D相关功能。

3.3.3 **virtio\_gpu\_get\_display\_info - 获取显示器信息**

|  |
| --- |
| C /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Name: virtio\_gpu\_get\_display\_info  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  static int virtio\_gpu\_get\_display\_info(FAR struct virtio\_gpu\_priv\_s \*priv) {  FAR struct virtqueue \*vq = priv->vdev->vrings\_info[VIRTIO\_GPU\_CTL].vq;  struct virtio\_gpu\_ctrl\_hdr cmd;  struct virtio\_gpu\_resp\_display\_info info;  struct virtqueue\_buf vb[2];  int ret;   memset(&cmd, 0, sizeof(cmd));  cmd.type = VIRTIO\_GPU\_CMD\_GET\_DISPLAY\_INFO;   vb[0].buf = &cmd;  vb[0].len = sizeof(cmd);  vb[1].buf = &info;  vb[1].len = sizeof(info);   ret = virtio\_gpu\_send\_cmd(vq, vb, 1, 1, NULL);  if (ret < 0)  {  return ret;  }   if (info.hdr.type != VIRTIO\_GPU\_RESP\_OK\_DISPLAY\_INFO)  {  return VIRTIO\_GPU\_MAP\_ERR(info.hdr.type);  }   priv->xres = info.pmodes[0].r.width;  priv->yres = info.pmodes[0].r.height;  vrtinfo("Setting resolution: (%d,%d)", priv->xres, priv->yres);  return OK; } |

3.3.4 **virtio\_gpu\_create\_2d - 在host上创建一个resource，获得资源id**

|  |
| --- |
| C /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Name: virtio\_gpu\_create\_2d  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  static int virtio\_gpu\_create\_2d(FAR struct virtio\_gpu\_priv\_s \*priv,  int resource\_id, int width, int height) {  FAR struct virtqueue \*vq = priv->vdev->vrings\_info[VIRTIO\_GPU\_CTL].vq;  struct virtio\_gpu\_resource\_create\_2d cmd;  struct virtio\_gpu\_ctrl\_hdr resp;  struct virtqueue\_buf vb[2];  int ret;   memset(&cmd, 0, sizeof(cmd));  cmd.hdr.type = VIRTIO\_GPU\_CMD\_RESOURCE\_CREATE\_2D;  cmd.resource\_id = resource\_id;  cmd.format = VIRTIO\_GPU\_FMT;  cmd.width = width;  cmd.height = height;   vb[0].buf = &cmd;  vb[0].len = sizeof(cmd);  vb[1].buf = &resp;  vb[1].len = sizeof(resp);   ret = virtio\_gpu\_send\_cmd(vq, vb, 1, 1, NULL);  if (ret >= 0 && resp.type != VIRTIO\_GPU\_RESP\_OK\_NODATA)  {  ret = VIRTIO\_GPU\_MAP\_ERR(resp.type);  }   return ret; } |

3.3.5 **virtio\_gpu\_attach\_backing - 给刚创建的resource连接guest(Vela)分配的(frame)buffer作为backing**

|  |
| --- |
| C /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Name: virtio\_gpu\_attach\_backing  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  static int virtio\_gpu\_attach\_backing(FAR struct virtio\_gpu\_priv\_s \*priv,  int resource\_id,  FAR struct virtio\_gpu\_mem\_entry \*ents,  uint32\_t nents) {  FAR struct virtqueue \*vq = priv->vdev->vrings\_info[VIRTIO\_GPU\_CTL].vq;  struct virtio\_gpu\_backing\_s backing;  struct virtio\_gpu\_ctrl\_hdr resp;  struct virtqueue\_buf vb[2];  size\_t i;  int ret;   if (nents > VIRTIO\_GPU\_MAX\_NENTS)  {  vrterr("ERROR: Backing memory entries count %d exceeds %d",  nents, VIRTIO\_GPU\_MAX\_NENTS);  return -E2BIG;  }   memset(&backing.cmd, 0, sizeof(backing.cmd));  backing.cmd.hdr.type = VIRTIO\_GPU\_CMD\_RESOURCE\_ATTACH\_BACKING;  backing.cmd.resource\_id = resource\_id;  backing.cmd.nr\_entries = nents;   for (i = 0; i < nents; i++)  {  backing.ents[i] = ents[i];  }   vb[0].buf = &backing;  vb[0].len = sizeof(backing.cmd) + nents \* sizeof(backing.ents[0]);  vb[1].buf = &resp;  vb[1].len = sizeof(resp);   ret = virtio\_gpu\_send\_cmd(vq, vb, 1, 1, NULL);  if (ret >= 0 && resp.type != VIRTIO\_GPU\_RESP\_OK\_NODATA)  {  ret = VIRTIO\_GPU\_MAP\_ERR(resp.type);  }   return ret; } |

3.3.6 **virtio\_gpu\_set\_scanout - 将resource与scanout（host端的输出）对接起来**

|  |
| --- |
| C /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Name: virtio\_gpu\_set\_scanout  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  static int virtio\_gpu\_set\_scanout(FAR struct virtio\_gpu\_priv\_s \*priv,  int scanout\_id, int resource\_id,  int width, int height) {  FAR struct virtqueue \*vq = priv->vdev->vrings\_info[VIRTIO\_GPU\_CTL].vq;  struct virtio\_gpu\_set\_scanout cmd;  struct virtio\_gpu\_ctrl\_hdr resp;  struct virtqueue\_buf vb[2];  int ret;   memset(&cmd, 0, sizeof(cmd));  cmd.hdr.type = VIRTIO\_GPU\_CMD\_SET\_SCANOUT;  cmd.scanout\_id = scanout\_id;  cmd.resource\_id = resource\_id;  cmd.r.width = width;  cmd.r.height = height;   vb[0].buf = &cmd;  vb[0].len = sizeof(cmd);  vb[1].buf = &resp;  vb[1].len = sizeof(resp);   ret = virtio\_gpu\_send\_cmd(vq, vb, 1, 1, NULL);  if (ret >= 0 && resp.type != VIRTIO\_GPU\_RESP\_OK\_NODATA)  {  ret = VIRTIO\_GPU\_MAP\_ERR(resp.type);  }   return ret; } |

3.3.7 **virtio\_gpu\_transfer\_to\_host\_2d - 将framebuffer的内容传输到host**

|  |
| --- |
| C /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Name: virtio\_gpu\_transfer\_to\_host\_2d  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  static int virtio\_gpu\_transfer\_to\_host\_2d(FAR struct virtio\_gpu\_priv\_s \*priv,  int resource\_id, int x, int y,  int width, int height) {  FAR struct virtqueue \*vq = priv->vdev->vrings\_info[VIRTIO\_GPU\_CTL].vq;  struct virtio\_gpu\_transfer\_to\_host\_2d cmd;  struct virtio\_gpu\_ctrl\_hdr resp;  struct virtqueue\_buf vb[2];  int ret;   memset(&cmd, 0, sizeof(cmd));   cmd.hdr.type = VIRTIO\_GPU\_CMD\_TRANSFER\_TO\_HOST\_2D;  cmd.resource\_id = resource\_id;  cmd.offset = y \* priv->stride + x \* (VIRTIO\_GPU\_BPP >> 3);  cmd.r.x = x;  cmd.r.y = y;  cmd.r.width = width;  cmd.r.height = height;   vb[0].buf = &cmd;  vb[0].len = sizeof(cmd);  vb[1].buf = &resp;  vb[1].len = sizeof(resp);   ret = virtio\_gpu\_send\_cmd(vq, vb, 1, 1, NULL);  if (ret >= 0 && resp.type != VIRTIO\_GPU\_RESP\_OK\_NODATA)  {  ret = VIRTIO\_GPU\_MAP\_ERR(resp.type);  }   return ret; } |

3.3.8 **virtio\_gpu\_flush\_resource - 通知host进行显示刷新**

|  |
| --- |
| C /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Name: virtio\_gpu\_flush\_resource  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  static int virtio\_gpu\_flush\_resource(FAR struct virtio\_gpu\_priv\_s \*priv,  int resource\_id, int x, int y,  int width, int height) {  FAR struct virtqueue \*vq = priv->vdev->vrings\_info[VIRTIO\_GPU\_CTL].vq;  FAR struct virtio\_gpu\_resource\_flush \*cmd;  struct virtqueue\_buf vb;   cmd = virtio\_zalloc\_buf(priv->vdev, sizeof(\*cmd), 16);  if (cmd == NULL)  {  vrterr("ERROR: Failed to allocate cmd buffer");  return -ENOMEM;  }   cmd->hdr.type = VIRTIO\_GPU\_CMD\_RESOURCE\_FLUSH;  cmd->resource\_id = resource\_id;  cmd->r.x = x;  cmd->r.y = y;  cmd->r.width = width;  cmd->r.height = height;   vb.buf = cmd;  vb.len = sizeof(\*cmd);   return virtio\_gpu\_send\_cmd(vq, &vb, 1, 0, cmd); } |

在QEMU上的配置和使用参考[VirtIO-GPU](https://xiaomi.f.mioffice.cn/wiki/wikk4RdMAWsV5vK7FuljZyhCFdg)。