Sst\_download\_fw \_-

1. Sst\_request\_firmware()
2. Sst\_load\_fw()

**If firmware not in memory call sst\_request\_fw()**

|  |
| --- |
| if (!sst\_drv\_ctx->fw\_in\_mem) {  retval = sst\_request\_fw(); /\* Requests audio fw from kernel and saves a copy  if (retval)  return retval;  } |

**Sst\_request\_fw()**

* This function requests the SST FW from the kernel, parses it and saves a copy in the driver context

|  |
| --- |
| snprintf(name, sizeof(name), "%s%04x", "fw\_sst\_", sst\_drv\_ctx->pci\_id);  pr\_debug("Requesting FW %s now...\n", name);  retval = request\_firmware(&sst\_drv\_ctx->fw, name, &sst\_drv\_ctx->pci->dev);  sst\_drv\_ctx->fw\_in\_mem = kzalloc(sst\_drv\_ctx->fw->size, GFP\_KERNEL);  memcpy(sst\_drv\_ctx->fw\_in\_mem, sst\_drv\_ctx->fw->data, sst\_drv\_ctx->fw->size);  sst\_parse\_fw\_image(sst\_drv\_ctx->fw\_in\_mem, sst\_drv\_ctx->fw->size,  &sst\_drv\_ctx->fw\_sg\_list) |

* sst\_parse\_fw\_image(const void \*sst\_fw\_in\_mem, unsigned long size, struct sst\_sg\_list \*sg\_list)

1. /\* Read the header information from the data pointer \*/

header = (struct fw\_header \*)sst\_fw\_in\_mem;

1. /\* verify FW \*/
2. module = (void \*)sst\_fw\_in\_mem + sizeof(\*header); /\*find the module\*/

|  |
| --- |
| for (count = 0; count < header->modules; count++) {  /\* for each module module \*/  ret\_val = sst\_parse\_module(module, sg\_list);  if (ret\_val)  return ret\_val;  module = (void \*)module + sizeof(\*module) + module->mod\_size ;  } |

* sst\_parse\_module(struct fw\_module\_header \*module, struct sst\_sg\_list \*sg\_list)

1. parse audio fw modules
2. Count the length for scattergather list
3. and create the scattergather list of same length

|  |
| --- |
| struct dma\_block\_info \*block;  unsigned long ram;  int retval, sg\_len = 0;  struct scatterlist \*sg\_src, \*sg\_dst;  block = (void \*)module + sizeof(\*module);  for (count = 0; count < module->blocks; count++) {  sg\_len += (block->size) / SST\_MAX\_DMA\_LEN; (4095\*4)  if ((block->size) % SST\_MAX\_DMA\_LEN)  sg\_len = sg\_len + 1;  block = (void \*)block + sizeof(\*block) + block->size;  }  sg\_src = kzalloc(sizeof(\*sg\_src)\*(sg\_len), GFP\_KERNEL)  sg\_init\_table(sg\_src, sg\_len);  sg\_dst = kzalloc(sizeof(\*sg\_dst)\*(sg\_len), GFP\_KERNEL);  sg\_init\_table(sg\_dst, sg\_len);  sg\_list->src = sg\_src;  sg\_list->dst = sg\_dst;  sg\_list->list\_len = sg\_len |

|  |
| --- |
| for (count = 0; count < module->blocks; count++) {  switch (block->type) {  case SST\_IRAM:  ram = sst\_drv\_ctx->iram\_base;  break;  case SST\_DRAM:  ram = sst\_drv\_ctx->dram\_base;  break;  /\*converting from physical to virtual because  scattergather list works on virtual pointers\*/  ram = (int) phys\_to\_virt(ram);  retval = sst\_fill\_sglist(ram, block, &sg\_src, &sg\_dst);  block = (void \*)block + sizeof(\*block) + block->size;  } |

* sst\_fill\_sglist(unsigned long ram, struct dma\_block\_info \*block, struct scatterlist \*\*sg\_src, struct scatterlist \*\*sg\_dst)

|  |
| --- |
| do {  dstn = (unsigned long)(ram + block->ram\_offset + offset);  src = (unsigned long)((void \*)block + sizeof(\*block) + offset);  len = block->size - offset;  pr\_debug("DMA blk src%lx,dstn %lx,size %d,offset %d\n",  src, dstn, len, offset);  if (len > SST\_MAX\_DMA\_LEN) {  pr\_debug("block size exceeds %d\n", SST\_MAX\_DMA\_LEN);  len = SST\_MAX\_DMA\_LEN;  offset += len;  } else {  offset = 0;  pr\_debug("Node length less that %d\n",  SST\_MAX\_DMA\_LEN);  }  if (!sg\_src || !sg\_dst)  return -ENOMEM;  sg\_set\_buf(\*sg\_src, (void \*)src, len);  sg\_set\_buf(\*sg\_dst, (void \*)dstn, len);  \*sg\_src = sg\_next(\*sg\_src);  \*sg\_dst = sg\_next(\*sg\_dst);  } while (offset > 0); |

**Sst\_load\_firmware**

* sst\_alloc\_dma\_chan(&sst\_drv\_ctx->dma); /\*alloc dma channel \*/

|  |
| --- |
| struct intel\_mid\_dma\_slave \*slave = &dma->slave;  dma\_cap\_zero(mask);  dma\_cap\_set(DMA\_MEMCPY, mask);  dma->dmac = pci\_get\_device(PCI\_VENDOR\_ID\_INTEL, PCI\_DMAC\_MFLD\_ID, NULL);  dma->ch = dma\_request\_channel(mask, chan\_filter, dma)  slave->dma\_slave.direction = DMA\_FROM\_DEVICE;  slave->hs\_mode = 0;  slave->cfg\_mode = LNW\_DMA\_MEM\_TO\_MEM;  slave->dma\_slave.src\_addr\_width = slave->dma\_slave.dst\_addr\_width =  DMA\_SLAVE\_BUSWIDTH\_4\_BYTES;  slave->dma\_slave.src\_maxburst = slave->dma\_slave.dst\_maxburst =  LNW\_DMA\_MSIZE\_16;  dmaengine\_slave\_config(dma->ch, &slave->dma\_slave); |

* sst\_dma\_firmware(struct sst\_dma \*dma, struct sst\_sg\_list \*sg\_list)

|  |
| --- |
| int retval = 0;  enum dma\_ctrl\_flags flag = DMA\_CTRL\_ACK;  struct scatterlist \*sg\_src\_list, \*sg\_dst\_list;  int length;  pr\_debug(" sst\_dma\_firmware\n");  sg\_src\_list = sg\_list->src;  sg\_dst\_list = sg\_list->dst;  length = sg\_list->list\_len;  sst\_drv\_ctx->desc = dma->ch->device->device\_prep\_dma\_sg(dma->ch,  sg\_dst\_list, length,  sg\_src\_list, length, flag);  if (!sst\_drv\_ctx->desc)  return -EFAULT;  sst\_drv\_ctx->desc->callback = sst\_dma\_transfer\_complete;  sst\_drv\_ctx->desc->callback\_param = sst\_drv\_ctx;  sst\_drv\_ctx->dma\_info\_blk.condition = false;  sst\_drv\_ctx->dma\_info\_blk.ret\_code = 0;  sst\_drv\_ctx->dma\_info\_blk.on = true;    sst\_drv\_ctx->desc->tx\_submit(sst\_drv\_ctx->desc);  retval = sst\_wait\_timeout(sst\_drv\_ctx, &sst\_drv\_ctx->dma\_info\_blk);  if (retval)  pr\_err("sst\_dma\_firmware..timeout!\n") |