Samsung-ipc compatible modems in Replicant

Denis 'GNUtoo' Carikli

July 28, 2019

Samsung IPC

Samsung IPC: Hardware part

XMM626 $\leftarrow$RAM$\rightarrow$SOC

- Galaxy S (I9100)
- Nexus S (I902x)
- Galaxy Tab (unsupported)

XMM626 ←RAM→SOC: Isolation

- Nexus S: The modem kernel driver shows that part of a RAM chip is shared between the modem and the SOC
- No hardware guarantees that the modem cannot take control of the SOC
- IOMMU:
  - Requires a mainline kernel to trust the code
  - And the SOC documentation on that...
  - And people having analyzed its security...
  - And to be setup before the RAM is even initialized...
  - But the bootloader is not free software...

XMM626 $\leftarrow$MIPI$\rightarrow$SOC

- Galaxy Nexus (I9250)
- Galaxy Tab II 7.0 (P3100)
- Galaxy Tab II 10.1 (P5100)

XMM626 ←MIPI→SOC: Isolation

- Not analyzed in depth but probably ok
- Same interface than camera and screens

XMM626 ←HSIC→SOC

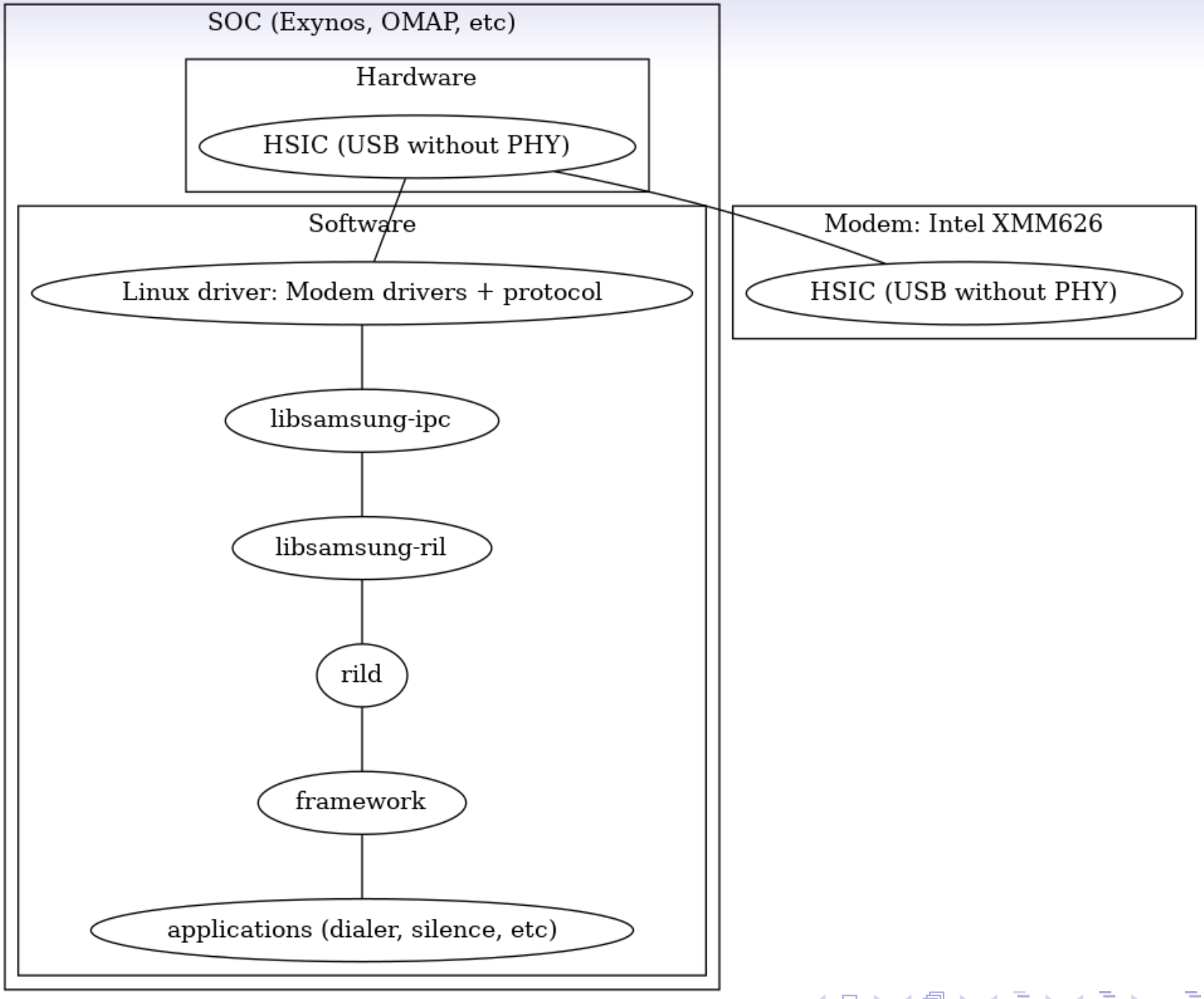
- Galaxy S2 (I9100)
- Galaxy SIII (I9300)
- Galaxy Note (N7000)
- Galaxy Note II (N7100)
- Galaxy Note 8.0 (N5100)

XMM626 ←HSIC→SOC: Isolation

- No DMA to the SOC RAM
- USB-like bus: USB without the PHY
- The host need to reset the bus to get the devices re-enumerated
- → More complicated for the modem to become a keyboard
- Ideally (not looked into yet):
  - usbguard
  - USB peripherals whitelist
  - → re-usable for USB modems (GTA04, PinePhone, Librem5, etc).
  - → Some modems do run GNU/Linux on one of their processor (Quectel EC25, etc)[1]. They can probably become a keyboard very easily.

Samsung IPC

- We will look more specifically at the case with the XMM626 connected through HSIC.
- Other transports are similar: transports are abstracted by the kernel driver and libsamsung-ipc transport drivers.


SOC (Exynos, OMAP, etc)
Hardware
HSIC (USB without PHY)
Software
Linux driver: Modem drivers + protocol
libsamsung-ipc
libsamsung-ril
rild
framework
applications (dialer, silence, etc)
Modem: Intel XMM626
HSIC (USB without PHY)

The samsung-ipc protocol

- Asyncronous
- $\rightarrow$ You get asyncronous responses from the modem
- $\rightarrow$ You match with the request (sequence numbers)
- $\rightarrow$ libsamsung-ril needs receive asyncronous responses too
- $\rightarrow$ More complex design with callbacks

libsamsung-ril

The initialization not very different from the reference-ril

```
RIL_RadioFunctions ril_radio_functions = {
    RIL_VERSION,
    ril_on_request,
    ril_on_state_request,
    ril_supports,
    ril_on_cancel,
    ril_get_version
};
...

const RIL_RadioFunctions *RIL_Init(
            const struct RIL_Env *env, int argc,
            char **argv) {
        ...
        return radio_functions;
}
```

The interface with rild is not that different either

```
    void ril_on_request(int request, void *data,
                        size_t size, RIL_Token token) {
    ...
    ril_request_register(request, data, size, token);
    ...
}
```

ril_request_register

```
int ril_request_register(int request, void *data,
                         size_t size, RIL_Token token) {
    ...
    list_end = ril_data->requests;
    ...
    list = list_head_alloc(list_end, NULL,
                           (void *) ril_request);
    ...
    ril_data->requests = list;
}
```

Global ril_Data variable

```
$ git grep "extern.*ril_data;"
samsung-ril.h:extern struct ril_data *ril_data;
```

Why?

- Asyncronous design $\rightarrow$ Faster
- Enqueue the request and continue serving new requests.
- We will see how getting the response works in a second time

- We got a request to power on the phone from the RIL
- We got rid of it by adding it to a list
- Now what happens to it?

RIL_Init

```
const RIL_RadioFunctions *RIL_Init(
            const struct RIL_Env *env, int argc,
            char **argv) {
    ...
    rc = pthread_create(&ril_data->request_thread,
                    &attr, ril_request_loop, NULL);
    ...
}
```

ril_request_loop

```
void *ril_request_loop(void *data) { ...
  while (1) {
    do {
      request = ril_request_find_status(
                                  RIL_REQUEST_UNHANDLED);
      ...
      request->status = RIL_REQUEST_PENDING;
    } while (request != NULL);
    do {
      ...
      request = ril_request_find_status(
                                  RIL_REQUEST_PENDING);
      ...
      rc = ril_request_dispatch(request);
    } while (request != NULL);
  }
```

ril_request_dispatch

```
int ril_request_dispatch(
struct ril_request *request) {
    ...
    for (i = 0; i < ril_request_handlers_count;
    i++) {
        ...
        if (ril_request_handlers[i].request ==
                                    request->request) {
            status = ril_request_handlers[i].handler(
                            quest->data, request->size,
                            request->token);
            ...
            request->status = status; ...
        } ...
    } ...
    return 0;
}
```

ril_request_handler

```
struct ril_request_handler
                                ril_request_handlers[] = {
    /* Power */
    {
        .request = RIL_REQUEST_RADIO_POWER,
        .handler = ril_request_radio_power,
    }, ...
}
```

ril_request_radio_power

```
int ril_request_radio_power(void *data,
size_t size, RIL_Token token) {
  ...
  power_state = *((int *)data);
  ...
  if (power_state > 0) {
    request_data.state =
        IPC_PWR_PHONE_STATE_REQUEST_NORMAL; ...
  } else {
    request_data.state =
        IPC_PWR_PHONE_STATE_REQUEST_LPM; ...
  } ...
  rc = ipc_fmt_send(ipc_fmt_request_seq(token),
        IPC_PWR_PHONE_STATE, IPC_TYPE_EXEC,
        (void *) &request_data,
        sizeof(request_data)); ...
}
```

ipc_fmt_send

```
int ipc_fmt_send(unsigned char mseq,
unsigned short command, unsigned char type,
const void *data, size_t size) {
    ...
    ipc_fmt_data = (struct ipc_fmt_data *)
                                    client->data;
    ...
    rc = ipc_client_send(ipc_fmt_data->ipc_client,
                mseq, command, type, data, size);
    ...
}
```

The rest happens in libsamsung-ipc

```
$ git grep IPC_PWR_PHONE_STATE_REQUEST_NORMAL
include/pwr.h:
#define IPC_PWR_PHONE_STATE_REQUEST_NORMAL 0x0202
$ git grep ipc_client_send
include/samsung-ipc.h:
int ipc_client_send(struct ipc_client *client, ...
samsung-ipc/ipc.c:
int ipc_client_send(struct ipc_client *client, ...
```

Now what happens with notifications from the modem?

Again RIL_Init

```
const RIL_RadioFunctions *RIL_Init(
const struct RIL_Env *env,
__attribute__((unused)) int argc,
__attribute__((unused)) char **argv) {
    ...
    rc = ril_client_loop(ril_clients[i]);
    ...
}
```

ril_clients[i]?

```
struct ril_client *ril_clients[] = {
        &ipc_fmt_client,
        &ipc_rfs_client,
        &srs_client,
};
```

IPC and RFS

- RFS: Modem's remote filesystem (EFS)
- SRS: For the audio part
- IPC: The rest of the protocol

Why do we have 3 different handlers

$\rightarrow$ Because of the kernel driver

```
$ git grep XMM626_SEC_MODEM_IPC0_DEVICE
modems/xmm626/xmm626_sec_modem.c:
fd = open(XMM626_SEC_MODEM_IPC0_DEVICE,
                    O_RDWR | O_NOCTTY | O_NONBLOCK);
modems/xmm626/xmm626_sec_modem.h:
#define XMM626_SEC_MODEM_IPC0_DEVICE
                                        "/dev/umts_ipc0"
$ git grep XMM626_SEC_MODEM_RFS0_DEVICE
modems/xmm626/xmm626_sec_modem.c:
fd = open(XMM626_SEC_MODEM_RFS0_DEVICE,
                    O_RDWR | O_NOCTTY | O_NONBLOCK);
modems/xmm626/xmm626_sec_modem.h:
#define XMM626_SEC_MODEM_RFS0_DEVICE
                                        "/dev/umts_rfs0"
```

ril_client_loop

```
int ril_client_loop(struct ril_client *client) {
    ...
    rc = pthread_create(&client->thread, &attr,
                        ril_client_thread, (void *) client);
    ...
}
```

ril_client_thread

```
void *ril_client_thread(void *data) {
    ...
    client = (struct ril_client *) data;
    ...
    rc = client->handlers->loop(client);
    ...
}
```

ril_client

```
struct ril_client ipc_fmt_client = {
        .id = RIL_CLIENT_IPC_FMT,
        .name = "IPC FMT",
        .handlers = &ipc_fmt_handlers,
        .callbacks = &ipc_fmt_callbacks,
};
```

ipc_fmt_handers

```
struct ril_client_handlers ipc_fmt_handlers = {
        .create = ipc_fmt_create,
        .destroy = ipc_fmt_destroy,
        .open = ipc_fmt_open,
        .close = ipc_fmt_close,
        .loop = ipc_fmt_loop,
};
```

ipc_fmt_loop

```
int ipc_fmt_loop(struct ril_client *client) {
    ...
    rc = ipc_client_recv(data->ipc_client,
                                        &message);
    ...
    rc = ipc_fmt_dispatch(client, &message);
    ...
}
```

ipc_fmt_dispatch

```
int ipc_fmt_dispatch(struct ril_client *client,
                     struct ipc_message *message) {
    ...
    for (i = 0;
      i < ipc_fmt_dispatch_handlers_count; i++) {
    ...
    if (ipc_fmt_dispatch_handlers[i].command ==
                                    message->command) {
    ...
    rc = ipc_fmt_dispatch_handlers[i].handler(
                                          message);
    ...
    }
  }
  ...
}
```

ipc_fmt_dispatch_handlers

```
struct ipc_dispatch_handler
                    ipc_fmt_dispatch_handlers[] = {
  /* Power */
  {
    .command = IPC_PWR_PHONE_PWR_UP,
    .handler = ipc_pwr_phone_pwr_up,
  },
```

ipc_pwr_phone_pwr_up

```
int ipc_pwr_phone_pwr_up(__attribute__((unused))
                         struct ipc_message *message) {
    ril_radio_state_update(RADIO_STATE_OFF);

    return 0;
}
```

libsamsung-ipc

ipc_client_send

```
int ipc_client_send(struct ipc_client *client,
unsigned char mseq, unsigned short command,
unsigned char type, const void *data,
                                    size_t size) {
    ...
    memset(&message, 0, sizeof(message));
    message.mseq = mseq;
    message.aseq = 0xff;
    message.command = command;
    message.type = type;
    message.data = (void *) data;
    message.size = size;

    return client->ops->send(client, &message);
}
```

client→ops→send

ipc_client_create

```
struct ipc_client *ipc_client_create(int type) {
    ...
    rc = ipc_device_detect();
    ...
    switch (type) {
        ...
        case IPC_CLIENT_TYPE_FMT:
            client->ops =
                        ipc_devices[device_index].fmt_ops;
            break;
            ...
    }
    ...
}
```

ipc_device_detect

```
int ipc_device_detect(void) {
    ...
    fd = open("/proc/cpuinfo", O_RDONLY);
    ...
    if (strncmp(line, "Hardware", 8) == 8) {
    ...
    if (...  && strcmp(kernel_version,
                ipc_devices[i].kernel_version) != 0)
    ...
}
```

ipc_devices

```
struct ipc_device_desc ipc_devices[] = {
    ...
    {
            .name = "i9300",
            .board_name = "smdk4x12",
            .kernel_version = NULL,
            .fmt_ops = &i9300_fmt_ops,
            .rfs_ops = &i9300_rfs_ops,
            .handlers = &i9300_handlers,
            .gprs_specs = &i9300_gprs_specs,
            .nv_data_specs = &i9300_nv_data_specs,
    },
```

ipc_client_ops

```
struct ipc_client_ops i9300_fmt_ops = {
        .boot = i9300_boot,
        .send = xmm626_sec_modem_fmt_send,
        .recv = xmm626_sec_modem_fmt_recv,
};
```

xmm626_sec_modem_fmt_send

```
int xmm626_sec_modem_fmt_send(
                        struct ipc_client *client,
                      struct ipc_message *message) {
    ...
    rc = client->handlers->write(
                    client->handlers->transport_data,
                                        p, length - count);
}
```

i9300_handlers

```
struct ipc_client_handlers i9300_handlers = {
    .read = i9300_read,
    .write = i9300_write,
    .open = i9300_open,
    .close = i9300_close,
    .poll = i9300_poll,
    .transport_data = NULL,
    .power_on = i9300_power_on,
    .power_off = i9300_power_off,
    .power_data = NULL,
    .gprs_activate = i9300_gprs_activate,
    .gprs_deactivate = i9300_gprs_deactivate,
    .gprs_data = NULL,
    .data_create = i9300_data_create,
    .data_destroy = i9300_data_destroy,
};
```

i9300_open

```
int i9300_open(void *data, int type)
{
    ...
    transport_data->fd =
                        xmm626_sec_modem_open(type);
    ...
    return 0;
}
```

xmm626_sec_modem_open

```
int xmm626_sec_modem_open(int type) {
    switch (type) {
      case IPC_CLIENT_TYPE_FMT:
        fd = open(XMM626_SEC_MODEM_IPC0_DEVICE,
                  O_RDWR | O_NOCTTY | O_NONBLOCK);
          break;
      case IPC_CLIENT_TYPE_RFS:
        fd = open(XMM626_SEC_MODEM_RFS0_DEVICE,
                  O_RDWR | O_NOCTTY | O_NONBLOCK);
          break;
      default:
          return -1;
    }
    ...
}
```

xmm626_sec_modem_fmt_send

```
$ git grep XMM626_SEC_MODEM_IPC0_DEVICE
devices/xmm626/xmm626_sec_modem.h:
#define XMM626_SEC_MODEM_IPC0_DEVICE "/dev/umts_ipc0"
$ git grep XMM626_SEC_MODEM_RFS0_DEVICE
devices/xmm626/xmm626_sec_modem.h:
#define XMM626_SEC_MODEM_RFS0_DEVICE "/dev/umts_rfs0"
```

i9300_write

```
int i9300_write(void *data,
        const void *buffer, size_t length) {
    ...
    rc = xmm626_sec_modem_write(
        transport_data->fd, buffer, length);

    return rc;
}
```

xmm626_modem_write

```
int xmm626_sec_modem_write(int fd,
                const void *buffer, size_t length) {
    ...
    status = ioctl(fd, IOCTL_MODEM_STATUS, 0);
    if (status != STATE_ONLINE &&
                                status != STATE_BOOTING)
        return -1;

    rc = write(fd, buffer, length);

    return rc;
}
```

Sharing the work with other Android distributions

- Not complete enough to be merged in LineageOS

Sharing the work with GNU/Linux distributions

- Freesmartphone.org
- Ofono + rild + libamsung-ril + libsamsung-ipc
- Ofono + libsamsung-ipc
- PostmarketOS?
- PureOS, Parabola?

Discussions:

- Sharing code with other distributions?
- USB Guard?
- Device requirements: Require modem to be isolated ?
- Other modems and GNU/Linux stack (PinePhone, GTA04)?
- Using GNU/Linux modem stack as much as possible?
- Automatic testing infrastructure and ofono?
- GNU/Linux and testing?

Licenses:

- `https://creativecommons.org/licenses/by-sa/4.0/`

See the following project for more details:
https://osmocom.org/projects/quectel-modems/wiki