Rev. 1, 8/2021

S32G 烧录工具基础：串口下载启动 A 核

by John.Li NXA08200; Tony.Zhang

当前 S32G 的 bootloader 镜像烧写一般可以通过 flash tool 来将 M7 镜像烧写到 QSPI NOR 中，或者 QSPI NOR 用烧录器烧写后，再贴装的方法。然后 bootloader 启动后再将其它镜像在线烧写到 eMMC 中，但是这样的办法有两个问题：

1. 工厂中需要增加一个 bootloader 烧写工位。
2. 先烧后贴的方式可能导致过炉时镜像损坏。

所以也有客户要求借助 Serial boot 功能实现直接在线烧录的方式。

当需要直接在线烧写 S32G 镜像时，基于以下两个事实：

- ROM Code 中 Serial boot 仅支持使用串口和 CAN 口下载。
- 快速下载方式除了网口外，也可以使用 USB 下载，因为 S32G 没有内置 USB PHY，需要硬件设计连接昂贵的 USB PHY，汽车应用中很少有此设计。

所以比较现实的方法是使用串口下载 uboot 到内部 RAM 中，启动 A53，然后在 uboot 驱动中实现了汽车级以太网 PHY 或交换机的驱动，再在 uboot 中连接到网口，使用网口来下载并烧录镜像。

但是 A 核并不能通过 serial boot 方法直接启动，所以需要使用 serial boot 通过串口下载一个 M7 镜像到内部 RAM 中，先让 M7 工程起来(Serial boot 只支持启动 M7 核)。然后再把 U-boot 与 M7 镜像交流，下载到内部 RAM 中，然后让 M7 去让 A53 启动 Uboot。

本文说明以上方式的实现，并作为烧录工具开发的一个基础。

版本	历史	作者
V1	● 创建本文	JohnLi

目录

1 开发重点

本工具的开发工作的重点和难点主要包括如下：

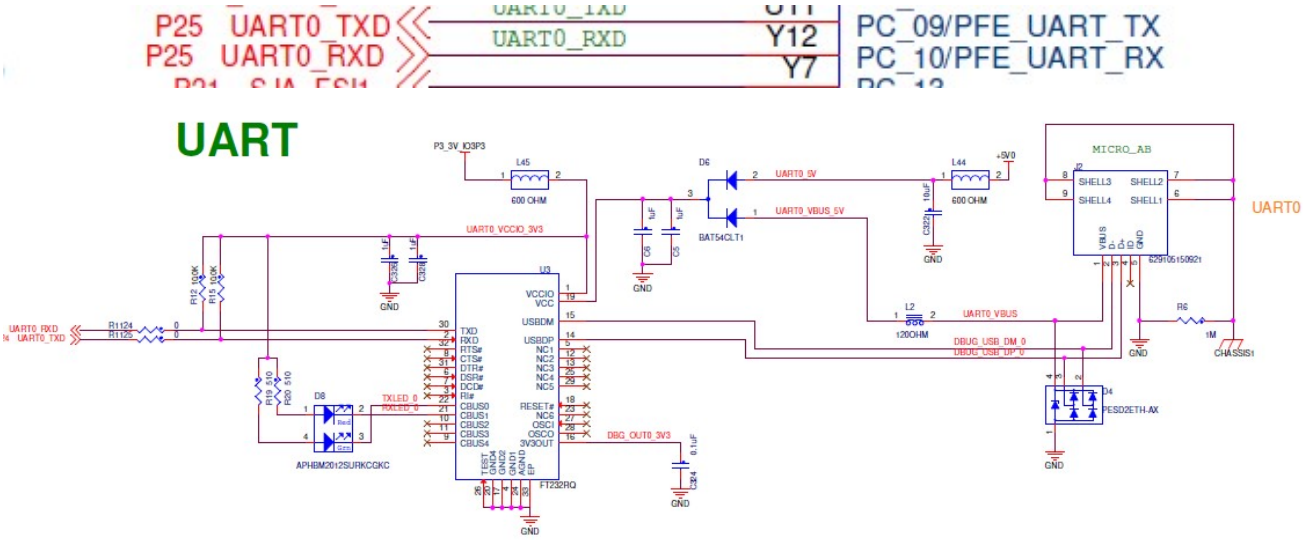
序号	开发工作	说明	开发者
1	开发	根据 S32G 的 serial boot 协议要求，开发 PC 端的串口工具来下载 M7 镜像	John.Li
2	开发	根据自定义协议要求，开发 PC 端的串口工具来下载 A 核 Bootloader 到 SRAM 中	John.Li
3	开发	根据自定义协议要求，开发 M7 镜像的串口接收与 Checksum 逻辑	John.Li
4	开发	修改 M7 镜像支持串口 0	John.Li
5	开发	开发实现 M7 镜像的串口单字节同步收发函数	John.Li
6	开发	开发实现 A53 启动功能	John.Li
7	调试与 Debug	调试解决串口接收乱码问题(Serial boot rom codes 仍然在回送消息串口)	John.Li
8	调试与 Debug	提供 解决 A 核启动串口 halt 思路(Serial boot rom codes 仍然占用串口)	John.Li
9	调试与 Debug	优化 M7 镜像，缩小大小	Tony.Zhang
10	调试与 Debug	根据 M7 镜像和 A 核 Uboot 在 SRAM 中的内存分配要求，重排 M7 镜像位置，避免冲突	Tony.Zhang
11	调试与 Debug	在 M7 中初始化 SRAM 空间	Tony.Zhang
12	调试与 Debug	在 M7 中设置 SRAM 可执行空间	Tony.Zhang
13	调试与 Debug	调试解决由于 cache 没有及时回写导致的下载镜像错误的问题	Tony.Zhang
14	调试与 Debug	集成，调优与文档	John.Li

所以总结一下本工具遇到并解决的主要重点与难点：

- 为了适配实现协议，开发了单字节同步访问串口函数，并调优。
- 根据 Debug，发现在 Serial boot 模式下，HSE 的 M7 会持续操作串口，这会导致两个问题，1 是在 PC 端 M7 串口回送数据时，会出现大量的数据，2 是在 uboot 启动时初始化串口时，会在退出 INIT 模式进入 Normal 模式时直接挂死，解决办法按照 serail boot 协议要求关掉 HSE M7 的串口操作。
- 因为移动过 M7 的内存分布，导致原来的 M7 MPU 寄存器设置不正确，导致不能执行代码，以及通过串口拷贝到内存中的 uboot 在内存中并不正确从而导致 uboot 执行失败(虽然串口数据的 checksum 正确)，这个是因为 cache 设置导致的，解决办法是重配 MPU 将目标内存修改成可执行，非 cacheable 的。

2 硬件说明

S32G RDB2 的串口连接如下：

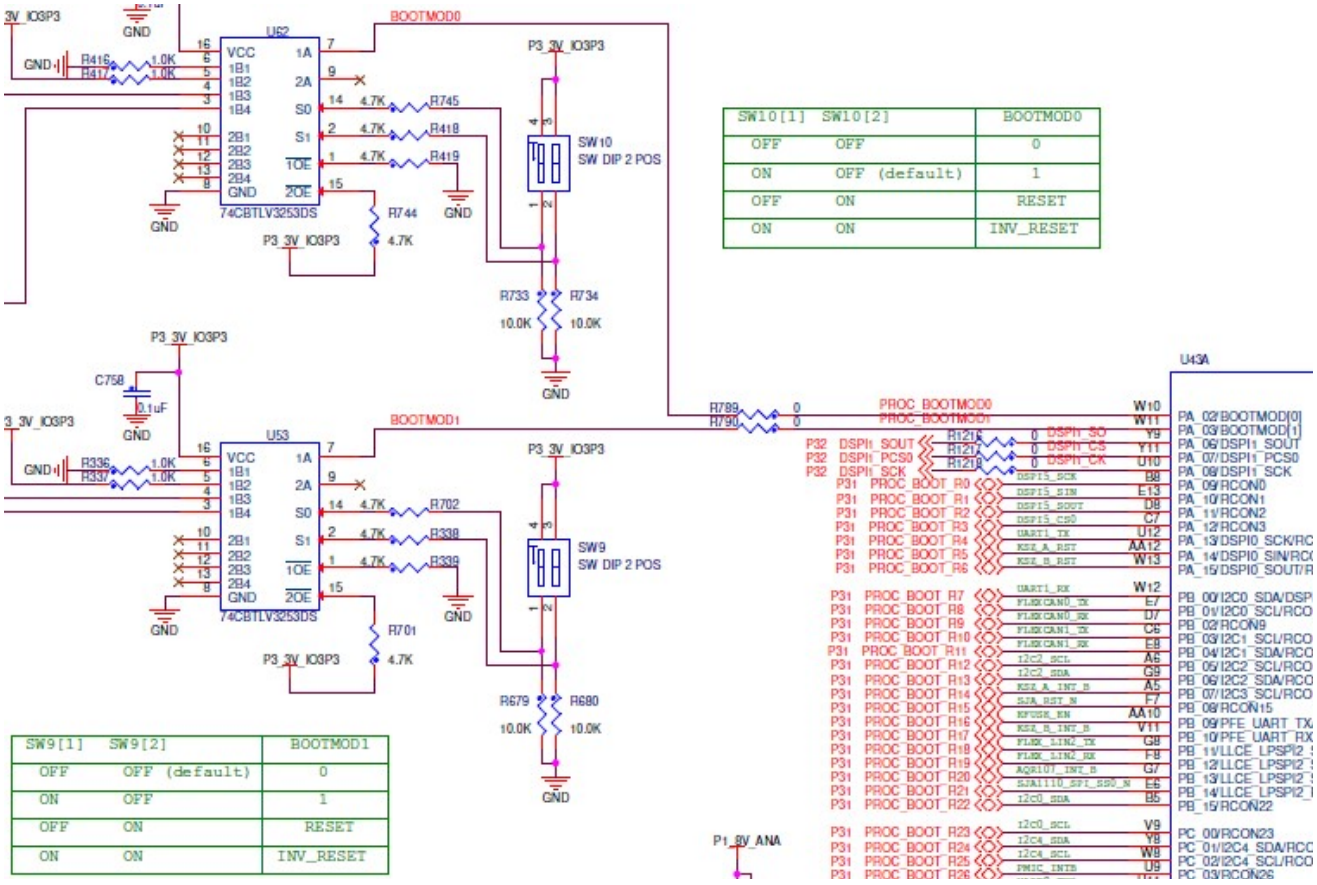


S32G RDB2 板设计的启动模式如下：

Table 132. Supported boot modes

Life cycle state	FUSE_SEL	BOOTMOD1	BOOTMOD2	Mode	Use case
MCU_PROD or CUST_DEL or OEM_PROD or IN_FIELD	0	0	0	Serial boot + XOSC in Differential mode	Production configuration for an unprogrammed chip
	0	0	1	Serial boot + XOSC in Crystal mode or Bypass mode	Production configuration for an unprogrammed chip
	0	1	0	Boot from RCON	Development configuration
	1	0	0	Boot from fuses	Production configuration for a programmed chip
	1	0	1	Boot from fuses	Production configuration for a programmed chip
	1	1	0	Serial boot	Debug of a programmed chip
	X	1	1	Reserved	Reserved

相应硬件设计为：



SW10[1]	SW10[2]	BOOTMOD0
OFF	OFF	0
ON	OFF (default)	1
OFF	ON	RESET
ON	ON	INV_RESET

SW9[1]	SW9[2]	BOOTMOD1
OFF	OFF (default)	0
ON	OFF	1
OFF	ON	RESET
ON	ON	INV_RESET

• **Serial Boot:**

– SW9 -> ALL-OFF

– SW10 -> ALL-OFF

SW9[1]	SW9[2]	BOOTMOD1
OFF	OFF (default)	0
ON	OFF	1
OFF	ON	RESET
ON	ON	INV_RESET

SW10[1]	SW10[2]	BOOTMOD0
OFF	OFF	0
ON	OFF (default)	1
OFF	ON	RESET
ON	ON	INV_RESET

3 所需工具和相关资料

文档：

	工具或文档	获得方式	说明
S32G 芯片手册	S32G2RM.pdf	https://www.nxp.com/s32g	参考 Boot->QSPI Nor boot 和 QSPI Nor 两章节
S32DS 帮助文档	S32DS->Help->Help Contents-> S32 Configuration Tools Getting Start->Pin	S32DS 连机文档	S32G Pin 工具说明
S32DS 帮助文档	S32DS->Help->Help Contents-> S32 Configuration Tools Getting Start->Peripherals	S32DS 连机文档	S32G Peripherals 工具说明
S32DS 帮助文档	S32DS->Help->Help Contents-> S32 Configuration Tools Getting Start->IVT Tool	S32DS 连机文档	S32G IVT 打包与烧写工具说明
RTD 驱动集成说明		\S32DS.3.4\S32DS\ software\ PlatformSDK_S32XX_2021_05\ SW32_RTD_4_4_2_0_0_D2105\ Uart_TS_T40D11M20I0R0\doc	
RTD 驱动使用说明			

工具：

- 参考文档《S32 Design Studio v3.4.1 及 RTD 2.0.0 HF3 安装指南_20210705.pdf》安装 S32 DS 3.4.1 和 RTD 2.0.0 HF3。
- Platform_Software_integration: Platform_Software_Integration_S32G2XX_2021_04.exe，需要借鉴之中的 bootloader 工程代码启动 A53 的方法。
- Visual Studio 2019，用于开发 PC 端 C#编写的串口工具。

4 Serial Boot 的启动流程

参考芯片手册：《S32G2RM.pdf》30.12 Serial boot

Serial Boot mode is entered via the BOOTMOD input pins. Serial download can also be initiated if the Functional Reset Counter(FREC register in the MC_RGM module) reaches a value ≥8. In Serial Boot mode, BootROM programs the HSE_H SWT for a 60-second timeout, then continuously polls for activity on any of the available interfaces:

• CAN
• UART

If no activity is detected, the timer expires and the core is reset. BootROM sequentially checks for activity on all available interfaces
and selects the first serial interface that it identifies as active as the download interface. The following figure describes the selection process.

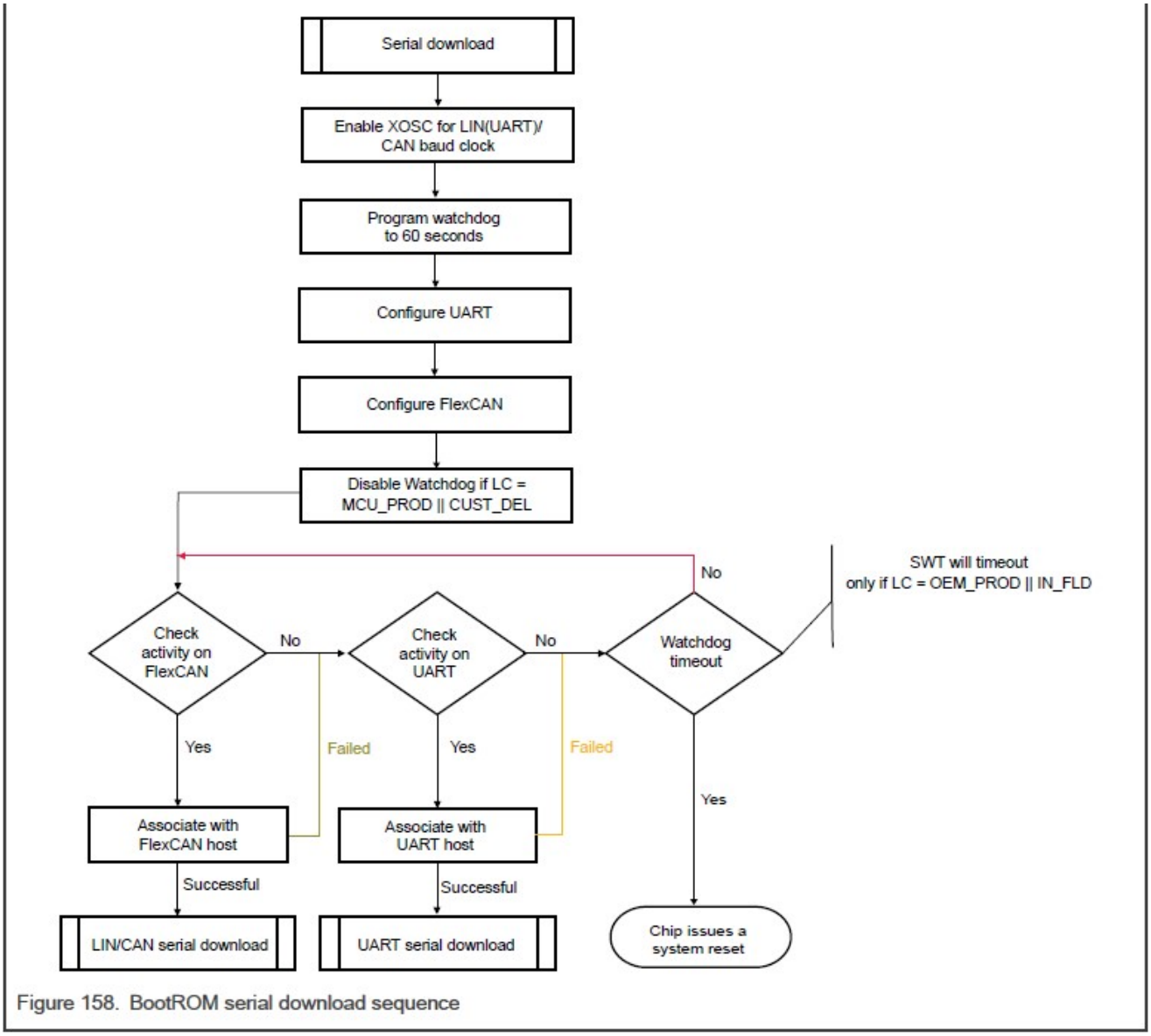


Figure 158. BootROM serial download sequence

下载协议如下：

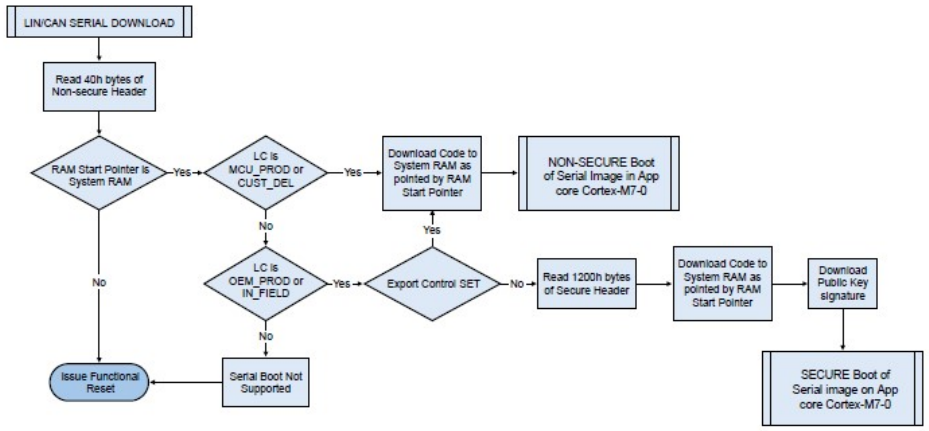


Figure 159. FlexCAN and UART download protocol

执行过程如下：

Table 163. Serial download protocol phases

No.	Phase	Description	Size (bytes)	Entry
1	Initialization	BootROM prepares the device to perform a serial download by performing the following steps: 1. Configure clock. 2. Configure the module. 3. Configure watchdog to 60 seconds. 4. Disable SWT if the life cycle state is MCU_PROD or CUST_DEL.	—	—
2	Association	BootROM polls sequentially for activity on the transmission channel until the marker FEED_FACE_CAFE_BEEFh is received. Any incorrect data received is discarded.	8h	Always
3	Non-secure header	BootROM reads the first 40h bytes over the channel and extracts the RAM start pointer, entry pointer, and the code size from the image. BootROM also extracts the NO_ECHO field from the most significant bit of the Code length word.	40h	Always

No.	Phase	Description	Size (bytes)	Entry
		NOTE • BootROM allows serial download of data only to system RAM. The maximum size of the serial image is limited to the size of the system RAM on the chip. • The image download address must be 64-bit aligned.		
4	Secure header	BootROM reads the next 1200h bytes of the image containing the NSK Public keys, CSK Public key, and the CSK signature.	1200h	Secure boot
5	Code download	BootROM downloads the serial code of specified length over the channel and stores it at the specified location.	Code length	Always
6	Public signature	BootROM reads the Public key signature of the image used for authentication of the image.	100h	Secure boot
7	Execution	After successful download of entire image, the Cortex-M7_0 core is enabled with the reset vector pointing to the Entry pointer. Before enabling the core, the watchdog SWT_0 is enabled with its default configuration. NOTE It is the application's responsibility to service the watchdog appropriately.	—	—

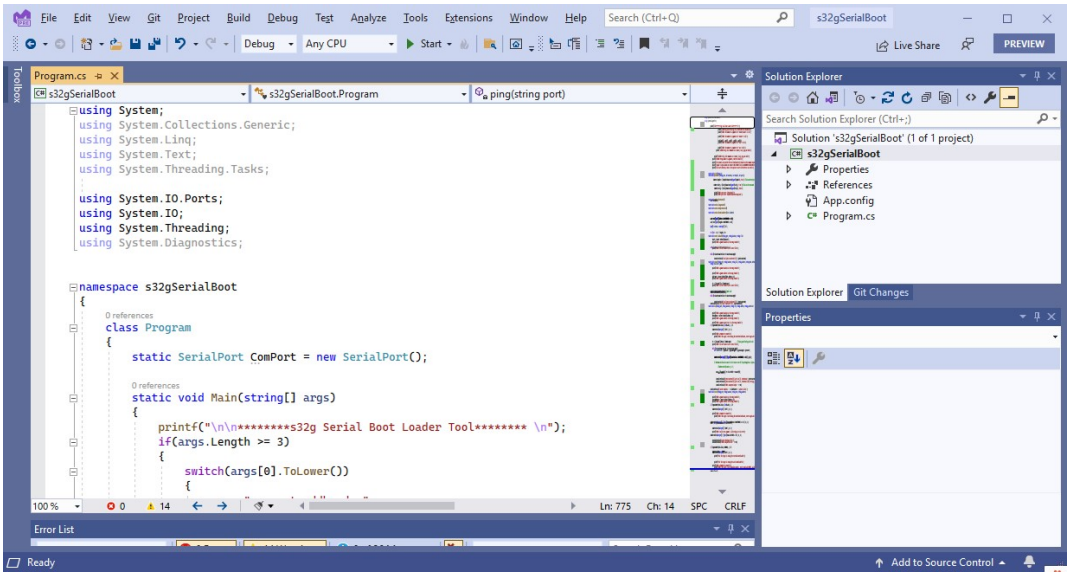
5 PC 端串口下载工具开发

5.1 开发环境搭建

PC 端的串口下载工具使用 C#开发，所以需要下载 Visual Stadio，可以从微软网站下载试用版：https://visualstudio.microsoft.com/->Download Visual Studio->Community 2019 社区版，下载安装包后，运行安装。

安装结束后，打开 s32gSerialBoot_Csharp-> s32gSerialBoot.sln，可能需要安装更多组件以支持编译，安装结束后，可以再打开工程。

打开后界面如下：



在代码区可以直接编辑，然后在 Build->Rebuild Solution 可以编译，编译结束后在：\s32gSerialBoot_Csharp\s32gSerialBoot\bin\Debug 可以看到编译成功的执行文件：s32gSerialBoot.exe。

源代码中使用 SerialPort 类来访问电脑串口，函数 openComPort 实现了对串口的配置：

```
ComPort.PortName = port;
 ComPort.BaudRate = baudrate;
ComPort.DataBits = 8;
 ComPort.StopBits = (StopBits)Enum.Parse(typeof(StopBits), "One");
 ComPort.Parity = (Parity)Enum.Parse(typeof(Parity), "Even");
```

而读写函数是 ComPort.Open();ComPort.Write(writeData, 0, 1); ComPort.ReadByte();ComPort.Close();

而主程序的核心是要实现串口的下载协议，包括两部分：

- 根据 S32G Serial Boot 的数据格式要求，组织并下载 M7 镜像到 S32G SRAM 中并运行起来。
- 自己定义协议，来把 Uboot 数据通过 PC 串口工具与内部 M7 镜像通讯，下载到 S32G SRAM 中，并通知 M7 去启动 A53 核。

5.2 利用 S32G Serial Boot 下载镜像到 SRAM 中并启动 M7 核

参考芯片手册关于 Serial Boot 头的要求：

2	Association	BootROM polls sequentially for activity on the transmission channel until the marker FEED_FACE_CAFE_BEEFh is received. Any incorrect data received is discarded.	8h	Always
3	Non-secure header	BootROM reads the first 40h bytes over the channel and extracts the RAM start pointer, entry pointer, and the code size from the image. BootROM also extracts the NO_ECHO field from the most significant bit of the Code length word.	40h	Always

Table 157. Image structure for non-secure serial boot

Address offset	Bytes	Name	Comments
0h	4	Image header	Marks start of serial boot code image (see Table 159).
4h	4	RAM start pointer	Pointer to first RAM address to which BootROM must load serial boot code.
8h	4	RAM Entry pointer	Pointer to out of RESET start of boot target core. For Cortex-M7, it corresponds to VTOR (Only Cortex-M7 is supported in Serial Boot).
Ch	4	Code length word	Length of code section of the image (see Table 160).
10h	48	Reserved	Reserved
40h	Code_len	Code	Code can be any size up to the maximum size of SRAM.

Table 159. Non-secure and secure boot image header

Byte 3	Byte 2	Byte 1	Byte 0
Version = 60h	Reserved	Reserved	TAG = D8h

The Code length word is described in the table below.

Table 160. Code length word

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NO_ECHO	CODE_SIZE														
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CODE_SIZE															

Table 161. Code length word field description

Field	Description
31 NO_ECHO	Do not echo incoming serial data from next serial phase onward.

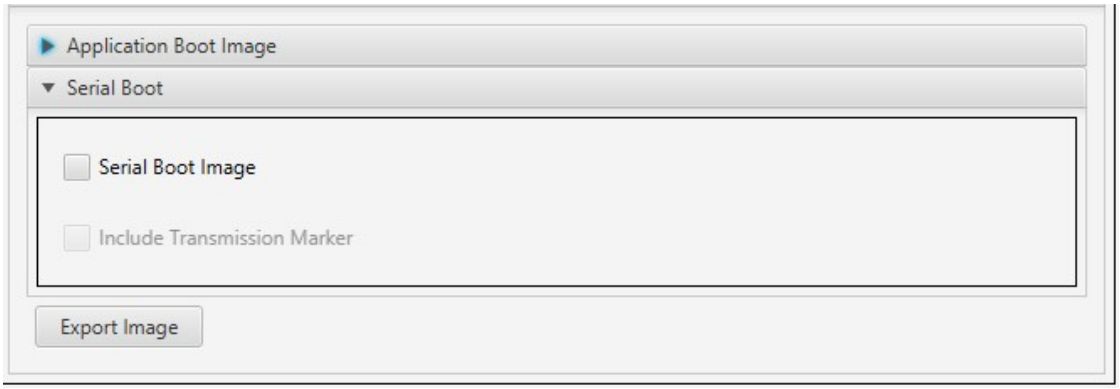
所以总结下来需要加在 Serial boot 的 M7 镜像头部的信息包括：

Header Marker	0xFEEDFACECAFEBEEF	8 Byte	
Image Header	0xD8000060	4 Byte	
Ram Start Point	小端(低地址在前)	4 Byte	从 M7 工程的连接文件中 int_sram 的地址获得
Ram Entry Point	小端(低地址在前)	4 Byte	从 M7 工程的 mapping 文件中宏 VTABLE 的地址获得
镜像长度	小端(低位在前)| 0x80000000	4 Byte	| 0x80000000 关掉 HSE M7 对串口的继续操作
Reserved	0xFF…	48 Byte	
实际镜像	M7 镜像 raw binary 文件	实际镜像大小	实际镜像大小不可以超过 SRAM 大小

以下为实际镜像示例：

所以只要在下载镜像中加上这个头，下载后就可以自动运行下载的 M7 镜像，准备头有两个方法：

1. 使用 S32DS 的 IVT 工具，在 Application Boot Code Image 栏中的 Serial Boot 中可以勾选 Serial Boot Image 和 Include Transmission Marker:



2. 另外一种方法就是为了不需要打包，可以在串口工具实现头部的发送字节，如下：

```
        sendSerialBootMarker();
        sendSerialImageHeader();
        sendSerialDownloadAddress(addr);
        sendSerialBootAddress(addr_boot);
        sendSerialBootLength((uint)br.BaseStream.Length);

        sendSerial48BytesReserved();

        //send image file
```

```
    static void sendSerialBootMarker()
    {
      uart_txrx(0xFE);
      uart_txrx(0xED);
      uart_txrx(0xFA);
      uart_txrx(0xCE);
      uart_txrx(0xCA);
      uart_txrx(0xFE);
      uart_txrx(0xBE);
      uart_txrx(0xEF);
        }
```

```
    static void sendSerialImageHeader()
    {
        uart_txrx(0xD8);
        uart_txrx(0x00);
        uart_txrx(0x00);
        uart_txrx(0x60);
    }
    static void sendSerialDownloadAddress(UInt32 address)
    {//小端在前
        uart_txrx((byte)(address & 0x000000ff));
        uart_txrx((byte)((address & 0x0000ff00) >> 8));
        uart_txrx((byte)((address & 0x00ff0000) >> 16));
        uart_txrx((byte)((address & 0xff000000) >> 24));
    }
    static void sendSerialBootAddress(UInt32 address)
    {//小端在前
        uart_txrx((byte)(address & 0x000000ff));
        uart_txrx((byte)((address & 0x0000ff00) >> 8));
        uart_txrx((byte)((address & 0x00ff0000) >> 16));
        uart_txrx((byte)((address & 0xff000000) >> 24));
    }
    static void sendSerialBootLength(uint length)
    {//小端在前
        length |= 0x80000000; //important!, which will disable the serial boot rom codes to disable the serial boot feedback
```

Field	Description
31 NO_ECHO	Do not echo incoming serial data from next serial phase onward.

```
        uart_txrx((byte)(length & 0x000000ff));
        uart_txrx((byte)((length & 0x0000ff00) >> 8));
        uart_txrx((byte)((length & 0x00ff0000) >> 16));
        uart_txrx((byte)((length & 0xff000000) >> 24));
    }
    static void sendSerial48BytesReserved()
    {
        // write bytes
        for (int i = 0; i < 48; i++)
        {
            uart_txrx(0xFF);
        }
}
```

其它主程序主要是实现配置串口，读文件并发送的功能：

```
static void connect(string port, string baudrate, string file, string address, string boot_address)
{…
if (!openComPort(serialLine, (int)baudr, 2, 1)) //注意一下 S32G 的 ROM code 对串口的要求是 48000， 8bit, 2bit 停止位和偶校验，特别是 2bit 停止位是与芯片手册不一致的地方，要小心
…
        while (br.BaseStream.Position < br.BaseStream.Length)
        {
           byte[] data = br.ReadBytes(64);
                        uart_txrx(data);
…
```

实际测试结果如第 7 章。

5.3 利用 S32G M7 串口驱动代码下载 Uboot 数据到 SRAM 中

可以利用 Serial boot 的机制，将一个串口测试的应用程序 M7 工程的 binary 下载到 SRAM 中运行，然后利用这个镜像来与 PC 端的串口继续通讯，把 Uboot 镜像下载到相应的 SRAM 中，这需要自己定义一下下载协议，如下代码：

```
static void load(string port, string baudrate, string file, string address, string packetsize)
    {…
    if (!openComPort(serialLine, (int)baudr, 1, 3)) //串口配置要与M7镜像一致，此处使用1bit停止位，无奇偶校验的方式
    …
            // marker for data
            ComPort.Write(new byte[] { 0x55 }, 0, 1); //发送0x55，表示要发送数据到SRAM的某个地址
            // address //发送地址
            ComPort.Write(new byte[] { (byte)((destAddress & 0x000000ff)) }, 0, 1);
            ComPort.Write(new byte[] { (byte)((destAddress & 0x0000ff00) >> 8) }, 0, 1);
            ComPort.Write(new byte[] { (byte)((destAddress & 0x00ff0000) >> 16) }, 0, 1);
            ComPort.Write(new byte[] { (byte)((destAddress & 0xff000000) >> 24) }, 0, 1);

            // length //发送Uboot镜像长度
            ComPort.Write(new byte[] { (byte)((transferSize & 0x000000ff)) }, 0, 1);
            ComPort.Write(new byte[] { (byte)((transferSize & 0x0000ff00) >> 8) }, 0, 1);
            ComPort.Write(new byte[] { (byte)((transferSize & 0x00ff0000) >> 16) }, 0, 1);
            ComPort.Write(new byte[] { (byte)((transferSize & 0xff000000) >> 24) }, 0, 1);
            //计算地址+长度数据的checksum
            sum += (int)(destAddress & 0xffff);
            sum += (int)((destAddress >> 16) & 0xffff);
            sum += (int)(transferSize & 0xffff);
            sum += (int)((transferSize >> 16) & 0xffff);
            while (sum > 0xffff) { sum -= 0xffff; }

            // send payload
            // checksum calculated as sum of 16 bit values so we will loop through data 2 bytes at a time
            byte[] transfer = new byte[2];
```

```
            // write bytes //发送文件
                ComPort.Write(data, 0, data.Length);
    //计算地址+尺寸+文件的checksum
    //发送checksum结果
    //监听串口，等待M7端传送送到数据的checksum校验结果，判断是否发送成功
```

程序运行成功后，可以用调试工具查看相应的下载地址内存内容，看看是否与下载的镜像一致。

PC端程序也会根据返回的checksum值打印是否下载成功。

5.4 利用 S32G M7 串口驱动代码启动 A53 在 SRAM 中的 Uboot

再次打开串口，根据自己定义的协议，通知 M7 将 A53 从某个地址启动，代码如下：

```
static void execute(string port, string baudrate, string core, string address)
    {…
    if (!openComPort(serialLine, (int)baudr, 1, 3)) //串口配置要与M7镜像一致，此处使用1bit停止位，无奇偶校验的方式
        // marker for data
        ComPort.Write(new byte[] { 0xaa }, 0, 1); //发送byte 0xaa，表示要通知M7去启动A53
        // address //发送启动地址
        ComPort.Write(new byte[] { (byte)((executeAddress & 0x000000ff)) }, 0, 1);
        ComPort.Write(new byte[] { (byte)((executeAddress & 0x0000ff00) >> 8) }, 0, 1);
        ComPort.Write(new byte[] { (byte)((executeAddress & 0x00ff0000) >> 16) }, 0, 1);
        ComPort.Write(new byte[] { (byte)((executeAddress & 0xff000000) >> 24) }, 0, 1);
        //依然做checksum
                sum += (int)(executeAddress & 0xffff);
        sum += (int)((executeAddress >> 16) & 0xffff);

        // send flags 表示要启动什么核
        if (core == "A53")
        {   // A53
            ComPort.Write(new byte[] { 0x01 }, 0, 1);
            ComPort.Write(new byte[] { 0x00 }, 0, 1);
            sum += 0x0001;
        }
        else if (core == "M7")
        {   // M7
            ComPort.Write(new byte[] { 0x00 }, 0, 1);
            ComPort.Write(new byte[] { 0x00 }, 0, 1);
            //sum += 0x0000;
        }
                // send checksum
        // listen for response //等待 M7 端回复是否校验 checksum 成功，表示是否执行命令成功。
```

最终测试结果见第 7 章。

5.5 实现下载启动一体化功能

为了提供效率，将 A 核 镜像下载和启动功能统一起来，并提供是否做 checksum 的选项，实现函数 load_to_run，其实质与前两章说明一致，请参数源代码。

6 S32G 端 M7 镜像开发

6.1 开发环境准备

由于要通过串口来下载 Uboot 镜像，所以我们可以基于 RTD 的串口示例代码来开发，首先，根据文档《S32 Design Studio v3.4.1 及 RTD 2.0.0 HF3 安装指南_20210705.pdf-TonyZhang》安装 S32DS v3.4.1+ RTD 2.0.0。

打开后 File->New->S32DS Project from Example 选中 Uart Examples->Linflexd_Uart_Ip_Example_S32G274A_M7。打开工程。

然后选中 ConfigTools->Peripherals 后，点击 Update Code 更新代码。

在 Project->Build Project.进行编译。

在工程上右击，然后 Debug As->Debug Configurations->Debugger,检查 Debug Probe Connection 是否连接上了 S32 Debug Probe, 或是直接点击 Debug 图标，即可以直接下载运行此示例程序。程序会从 UART1 打印出串口消息，并等待输入(默认串口设置是 96000, 8bit ,1bit 停止位，无奇偶检验，无流控)。

6.2 串口从 1 改成 0

此示例程序默认使用串口 1，而 Serial port 使用串口 0，所以我们先要把串口改成串口 0.需要修改 Pin 和串口驱动设置，及驱动调用。

1. ConfigTools->Pin->Routing Details 中，把已经有的 UART1 的 Pins 删除掉，增加：
   - LINFlexD_0 rxd [Y12]PC_10
   - LINFlexD_0 txd [U11]PC_09
2. ConfigTools->Peripherals->Components->Drivers->Linflexd_Uart_1->UartGlobalConfig->UartHwChannel 改成 LinflexD_0
3. 主程序 Main 函数在初始化时，需要手工修改

```
IntCtrl_Ip_EnableIrq(LINFLEXD0_IRQn); //1改0
IntCtrl_Ip_InstallHandler(LINFLEXD0_IRQn, LINFLEXD0_UART_IRQHandler, NULL_PTR); /1改0
..
```

```
  /* Initialize LINFlexD module for UART usage */
  Linflexd_Uart_Ip_Init(LINFLEXD_INSTANCE, &Linflexd_Uart_Ip_pHwConfigPB_0_VS_0); /1 改 0
…
Linflexd_Uart_Ip_AsyncSend/Linflexd_Uart_Ip_AsyncReceive 调用句柄
#define LINFLEXD_INSTANCE 0U //johnli 1 改 0
```

以上修改后，再运行程序就可以使用串口 0 来收发数据了。

另外本程序为一个示例程序，修改成下载工具后，原来的串口测试代码可以不需要了，所以在串口相关的初始化完成后，可以注掉其应用主程序。

然后程序 主体调用改成：

```
Bootloader_listenForPackets();//johli add for main function to enable the download protocol
```

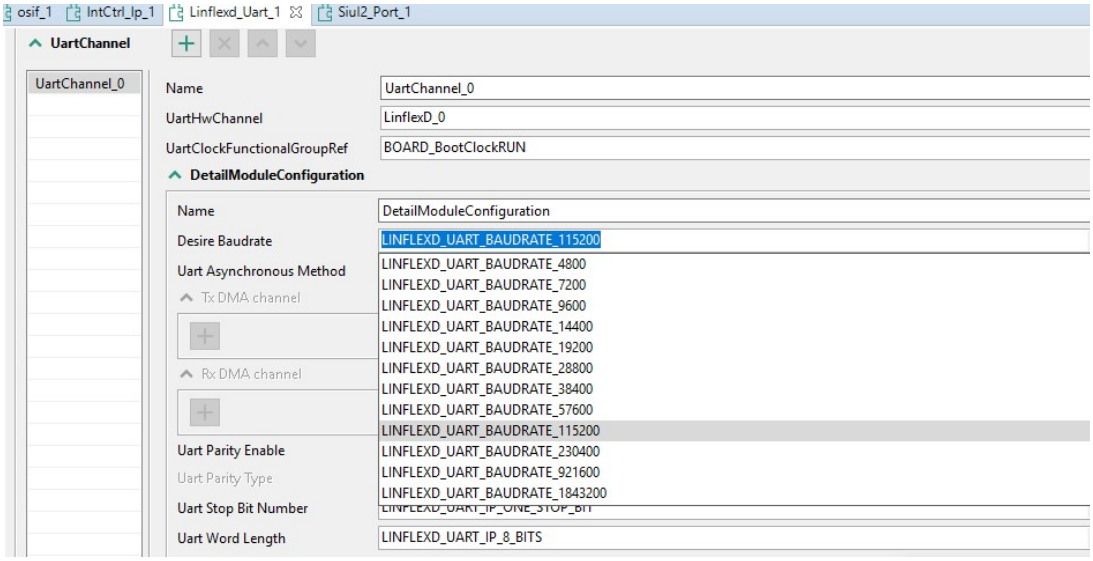
这个函数实现从串口接收一个 BYTE，根据这个 BYTE 判断是接收数据还是执行启动命令：

```
while(1) {
uint8_t receivedByte = in_char();
if (receivedByte == PACKET_DATA){
        receivePacketData();
} else if (receivedByte == PACKET_EXECUTE){
        receivePacketExecute();
        break;
}
```

6.3 串口速率修改

本示例程序默认是 96000 波特率，为了提高下载效率，可以适当增大，此处先测试 115200 波特率。

ConfigTools->Peripherals->Components->Drivers->Linflexd_Uart_1->Desire Buadrate 改成 LINFLEXD_UART)BAUDRAE_115200



6.4 实现串口的同步单字节访问收发函数

RTD 的串口示例程序是一个以中断方式访问串口的方法，本工具是以每一个 BYTE 来传输的，修改为非中断 Polling 的方式来提高 效率：

首先，将中断相关代码注掉：

main.c: function main:

```
// IntCtrl_Ip_EnableIrq(LINFLEXD0_IRQn); //johli disable interrupt
// IntCtrl_Ip_InstallHandler(LINFLEXD0_IRQn, LINFLEXD0_UART_IRQHandler, NULL_PTR); //johli disable interrupt
```

只保留时钟,iomux 与串口初始化：

```
Clock_Ip_Init(&Mcu_aClockConfigPB[0]);
Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfigArr0);
Linflexd_Uart_Ip_Init(LINFLEXD_INSTANCE, &Linflexd_Uart_Ip_pHwConfigPB_0_VS_0);
```

然后，对串口调用 API 做一个包装，使用 Polling 的方式来读写一个 BYTE 的串口数据：

```
char in_char (void)
{
    uint8 rxBuff=0x00;
    LINFLEXD_Type * pBase;
    pBase = Linflexd_Uart_Ip_apBases[LINFLEXD_INSTANCE];

  /* enable the receiver */
  LINFLEXD_SetReceiverState(pBase, TRUE);
  /* Wait until data reception flag is set or during reception */
  while (!LINFLEXD_GetStatusFlag(pBase, LINFLEXD_UART_DATA_RECEPTION_COMPLETE_FLAG))
  { ASM_KEYWORD(" nop "); }
    /* Retrieve the data */
```

```
        rxBuff=LINFLEXD_GetRxDataBuffer1Byte(pBase);
    /* Clear the flags */
    LINFLEXD_ClearStatusFlag(pBase, LINFLEXD_UART_DATA_RECEPTION_COMPLETE_FLAG);
    LINFLEXD_ClearStatusFlag(pBase, LINFLEXD_UART_MESSAGE_BUFFER_FULL_FLAG);
  /* disable the receiver */
  // LINFLEXD_SetReceiverState(pBase, FALSE); //由于大部分时间是在接收，实际应用中发现一直打开接收功能 可能提高稳定性
        return(rxBuff);
}

void out_char (char ch)
{
        LINFLEXD_Type * pBase;
        pBase = Linflexd_Uart_Ip_apBases[LINFLEXD_INSTANCE];

      /* Enable the transmitter */
      LINFLEXD_SetTransmitterState(pBase, TRUE);
        LINFLEXD_ClearStatusFlag(pBase, LINFLEXD_UART_DATA_TRANSMITTED_FLAG);
        LINFLEXD_SetTxDataBuffer1Byte(pBase, ch);
        /* Wait until data transmited flag is set during transmission */
        while (!LINFLEXD_GetStatusFlag(pBase, LINFLEXD_UART_DATA_TRANSMITTED_FLAG))
          { ASM_KEYWORD(" nop "); }
      LINFLEXD_ClearStatusFlag(pBase, LINFLEXD_UART_DATA_TRANSMITTED_FLAG);
      /* Disable the transmitter */
      LINFLEXD_SetTransmitterState(pBase, FALSE);
}
```

6.5 实现数据下载协议

结合 4.3 节 PC 端工具的做法，实现协议：

先接受镜像地址，并计算 checksum：

```
        // get address
        uint32_t address = 0;
        address += in_char();
        address += (in_char() << 8);
        address += (in_char() << 16);
        address += (in_char() << 24);
        sum = WRAP( (address) & (0xffff));
        sum = WRAP( sum + ((address >> 16) & 0xffff));
```

再接受镜像大小，并计算 checksum：

```
        // get length
        uint32_t length = 0;
        length += in_char();
        length += (in_char() << 8);
        length += (in_char() << 16);
```

```
length += (in_char() << 24);
sum = WRAP( sum + (length & 0xffff));
sum = WRAP( sum + ((length >> 16) & 0xffff));
```

再按照 16bit 来把数据接收到地址处，地址每接收一个 16bit 数据，就加 2，再计算 checksum:

```
// copy payload
// calculate checksum along the way
while(length > 0){
        if (length > 1){
                first = in_char();
                second = in_char();
                data = (second << 8) + first;
                length -= 2;
        } else {
                first = in_char();
                data = first;
                length -= 1;
        }
        *(uint16_t *) address = data;
        address += 2;
        sum = WRAP(sum + data);
}
```

然后再接收并比较 checksum:

```
// receive and compare checksum
uint8_t csumLower = in_char();
uint8_t csumUpper = in_char();
uint16_t csum = (csumUpper << 8) + csumLower;
sum = WRAP(sum + csum);
```

最后，根据 checksum 结果，回送下载结果：

```
// send response
if (sum == 0xFFFF) {
        // everything ok with transmission
        out_char(0x00);
} else {
        // transmission error
        out_char(0xFF);
}
```

6.6 实现启动 A53 方法

结合 4.4 节 PC 端工具的做法，实现协议：

先接受启动地址，并计算 checksum：

```
// get address
uint32_t address = 0;
address += in_char();
address += (in_char() << 8);
```

```
        address += (in_char() << 16);
        address += (in_char() << 24);
        sum = WRAP( (address) & (0xffff));
        sum = WRAP( sum + ((address >> 16) & 0xffff));
```

再接收标志位，并计算 checksum，此标志位标志是否启动 A53 或 M7。

```
        // get flags
        uint8_t flagsLower = in_char();
        uint8_t flagsUpper = in_char();
        uint16_t flags = (flagsUpper << 8) + flagsLower;
        sum = WRAP(sum + flags);
```

再比较发送过来的 checksum 比较是否协议发送成功：

```
        // receive and compare checksum
        uint8_t csumLower = in_char();
        uint8_t csumUpper = in_char();
        uint16_t csum = (csumUpper << 8) + csumLower;
        sum = WRAP(sum + csum);
```

回送发送是否成功的标志给 PC:

```
        // send response
        if (sum == 0xFFFF) {
                // everything ok with transmission
                out_char(0x33);
        } else {
                // transmission error
                out_char(0x99);
        }
```

以下，如果收到的启动标志是 1，表示要从 A53 启动，从芯片手册上的说明：

28.12.4 Software reset partition turn-on flowchart

For the Cortex-A53 reset domain, clocks for the initial frequencies can be set up by configuring MC_CGM_1 to the FIRC clock. See the clocking chapter for details.

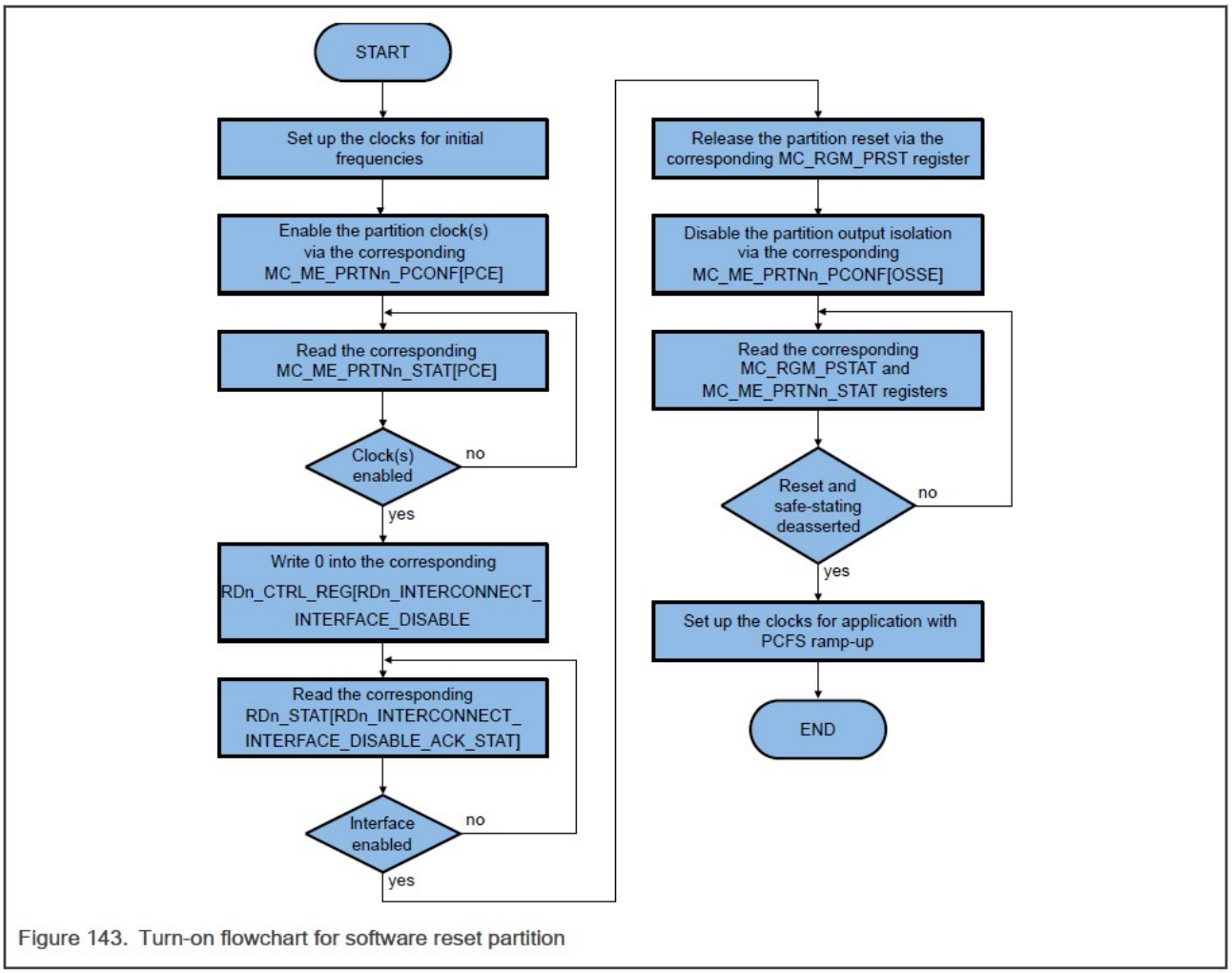


Figure 143. Turn-on flowchart for software reset partition

以下代码是 M7 启动 A53 的方法，此方法要从 Bootloader 工程中 porting 过来，目前安装的 Platform_software_Integration 的版本是：Platform_Software_Integration_S32G2XX_2021_04.exe

安装结束后，其 M7 启动 A53 的代码是：

```
/*==========================================================================================
==========*/
/**
* @brief   Starts an application
* @details Starts an application based on the input application configuration
* @param[in]     applicationConfig The application configuration.
```

```
* return        E_OK if success
*               E_NOT_OK if error
*/
/*==========================================================================================
=============*/
uint8_t Bl_StartApplication(Bl_ApplicationDetails * applicationConfig)
{
…
   if (E_OK == Bl_EnablePartition(partition))
   {
  …
          status = E_OK;
   }

   return status;
}
```

简化代码如下：

```
int EnableA53_0(int coreStartAddress)
{
   /* Keep track of the time needed to start the partition.
      If it takes too much time, consider the partition disabled */
          uint16_t timeout = TIMEOUT_CNT_INIT_VALUE;
   /* Auxiliary variable used for R/W access to registers */
   uint32_t regValue = 0;
   uint8_t status = E_NOT_OK;
   uint32_t resetCore;

   int partition=1; //A53 is partition 1
   int coreId=0; //first A53
   /* Set the core Vector Table address before enabling the partition */
   REG_WRITE32 (MC_ME_PRTN_N_CORE_M_ADDR(partition, coreId), coreStartAddress);

//enable partition:
   /* enable clock partition */
   REG_WRITE32 (MC_ME_PRTN_N_PCONF(partition), MC_ME_PARTITION_CLOCK_ENABLE);

   /* trigger hardware process for enabling clocks */
   REG_WRITE32 (MC_ME_PRTN_N_PUPD(partition), MC_ME_TRIGGER_PROCESS);

   /* write the valid key sequence */
   REG_WRITE32 (MC_ME_CTL_KEY_ADDR, MC_ME_CTL_KEY_VALUE);
   REG_WRITE32 (MC_ME_CTL_KEY_ADDR, MC_ME_CTL_KEY_INVERTED_VALUE);
```

```
    /* wait for partition clock status bit */
    while (((REG_READ32(MC_ME_PRTN_N_STAT(partition)) & MC_ME_PARTITION_CLOCK_ENABLE) != 1)
&& (timeout--));

    /* unlock software reset domain control register */
    regValue = REG_READ32(RDC_RD1_CTRL_REGISTER) | RDC_RD1_CTRL_UNLOCK_ENABLE;
    REG_WRITE32(RDC_RD1_CTRL_REGISTER, regValue);

    /* enable the interconnect interface of software reset domain */
    regValue = REG_READ32(RDC_RD1_CTRL_REGISTER) & RDC_RD1_XBAR_INTERFACE_DISABLE;
    REG_WRITE32(RDC_RD1_CTRL_REGISTER, regValue);

    /* wait for software reset domain status register
       to acknowledge interconnect interface not disabled */
    while((((REG_READ32(RDC_RD1_STAT_REGISTER)) & RDC_RD1_XBAR_INTERFACE_STAT) != E_OK)
&& (timeout--));

    /* cluster reset */
    regValue = REG_READ32(RGM_PRST(partition)) & RGM_PRST_CLUSTER;
    REG_WRITE32 (RGM_PRST(partition), regValue);

    regValue = REG_READ32(MC_ME_PRTN_N_PCONF(partition)) & MC_ME_OUTPUT_PARTITION;
    REG_WRITE32 (MC_ME_PRTN_N_PCONF(partition), regValue);

    regValue = REG_READ32(MC_ME_PRTN_N_PUPD(partition)) | MC_ME_OUTPUT_STATUS;
    REG_WRITE32 (MC_ME_PRTN_N_PUPD(partition), regValue);

    /* write the valid key sequence */
    REG_WRITE32 (MC_ME_CTL_KEY_ADDR, MC_ME_CTL_KEY_VALUE);
    REG_WRITE32 (MC_ME_CTL_KEY_ADDR, MC_ME_CTL_KEY_INVERTED_VALUE);

    /* wait until cluster is not in reset */
    while ((REG_READ32(RGM_PSTAT(partition)) & RGM_PSTAT_RESET_STATE) != E_OK);
    while (((REG_READ32(MC_ME_PRTN_N_STAT(partition)) & MC_ME_OUTPUT_STATUS) != 0x0) &&
(timeout--));

    /* lock the reset domain controller */
    regValue = REG_READ32(RDC_RD1_CTRL_REGISTER) & RDC_RD1_CTRL_UNLOCK_DISABLE;
    REG_WRITE32(RDC_RD1_CTRL_REGISTER, regValue);

    if (TIMEOUT_CNT_END_VALUE != timeout)
    {
       status = E_OK;
    }
    else
    {
          return(status);
    }
//end enable partitio
```

```
    /* enable core clock */
    REG_WRITE32 (MC_ME_PRTN_N_CORE_M_PCONF(partition, coreId), 1);

    /* Partition peripherals are always enabled in partition 0 */
    if (0 != partition) //PARTITION_0=0
    {
        REG_WRITE32 (MC_ME_PRTN_N_PCONF(partition), 1);
    }
    /* trigger hardware process */
    REG_WRITE32 (MC_ME_PRTN_N_CORE_M_PUPD(partition, coreId), 1);

    /* write key sequence */
    REG_WRITE32 (MC_ME_CTL_KEY_ADDR, MC_ME_CTL_KEY_VALUE);
    REG_WRITE32 (MC_ME_CTL_KEY_ADDR, MC_ME_CTL_KEY_INVERTED_VALUE);

    /* wait for clock to be enabled */
    /* @Todo: Move to SysDAL routines the wait for clock and core cluster reset. Call them here through API */
    while (((REG_READ32(MC_ME_PRTN_N_CORE_M_STAT(partition, coreId)) &&
MC_ME_CORE_CLOCK_STAT_MASK) != 1) && (timeout--)) ;
    while (((REG_READ32(MC_ME_PRTN_N_STAT(partition)) & MC_ME_CLOCK_ACTIVE) != 1) &&
(timeout--));

    /* pull the core out of reset and wait for it */
    resetCore = REG_READ32(RGM_PRST(partition)) & RGM_PRST_RESET_CORE(partition, coreId);
    REG_WRITE32 (RGM_PRST(partition), resetCore);
    while ((REG_READ32(RGM_PSTAT(partition)) & RGM_PRST_STATUS_CORE(partition, coreId)) != 0);

    if (TIMEOUT_CNT_END_VALUE != timeout)
    {
        status = E_OK;
    }
    else
    {
            return(status);
    }
}
```

最后 receivePacketExecute 函数中调用此函数启动 A53：

```
            } else if (flags == 0x0001) {
                        // bring up A53 Core 0
                        EnableA53_0(address);

            }
```

为了避免 uboot 在 53 时钟初始化和串口初始化与 M7 的代码冲突，在运行 A53 前先把串口的 clk 反初始化：

```
    Linflexd_Uart_Ip_Deinit(LINFLEXD_INSTANCE); //johnli li disable uart .
```

时钟的反初始化函数需要把本地静态连接的函数暴露出来：

RTD/src/Clock_Ip.c
// static void ResetClockConfiguration(Clock_Ip_ClockConfigType const * config)

void ResetClockConfiguration(Clock_Ip_ClockConfigType const * config) //johnli modify

RTD/include/Clock_Ip.h

void ResetClockConfiguration(Clock_Ip_ClockConfigType const * config); //johnli add

src/main.c
  ResetClockConfiguration(&Mcu_aClockConfigPB[0]); //johnli reset clock

  EnableA53_0(address_run);

之后工程的代码 Update Code 时注意不要覆盖。

6.7 实现下载启动一体化功能

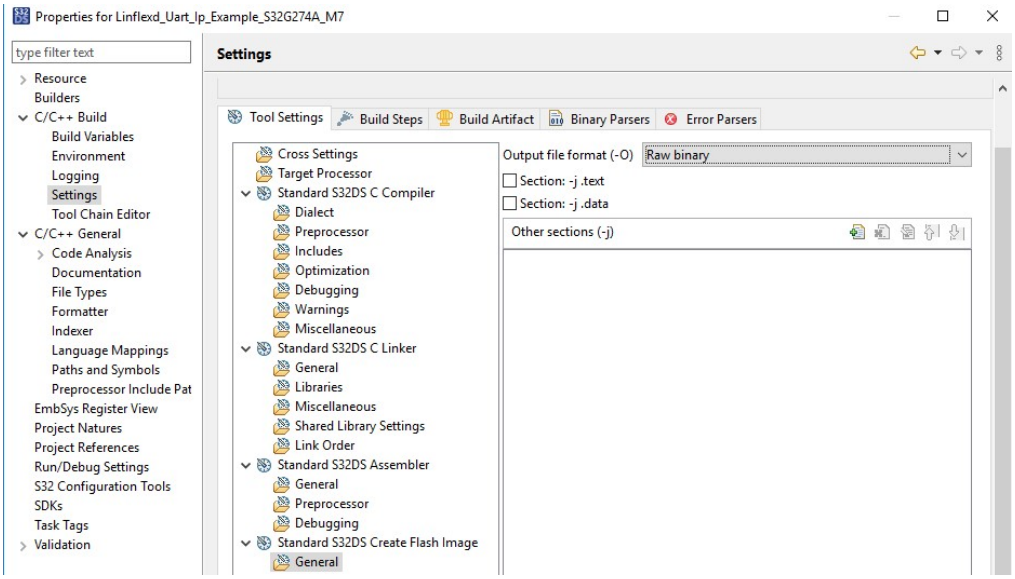
为了提供效率，将 A 核 镜像下载和启动功能统一起来，并提供是否做 checksum 的选项，实现函数 receivePacketData_to_run，其实质与前两章说明一致，请参数源代码。

6.8 优化镜像大小及避免与 Uboot 镜像冲突

串口下载要求使用 raw binary 文件，而 S32DS 默认只是编译出 elf 文件用于调试器下载,所以首先要配置输出 bin 文件：

Project->Properities->C/C++ Build->Settings->勾选 Create flash image，Apply and Close 后，就会出现：Standard S3DS Create Flash Image->

General 中把 Output file format(-O)改成 Raw binary:



这样就可以编译出*.bin 文件了，但是默认编译出来的镜像大小是 5.377KB，对应的 elf 反到只有 1.754KB，所以这样的 binary 用串口下载影响效率(ROM codes 默认 serial boot 波特率只有 48000)。

所以参考文档《02. S32G FAQ_v20210713.pdf-TonyZhang》1.3 节：RTD 工程的链接文件的介绍

- 样例工程的链接文件只是一个示例，用户可根据自己的需求对链接文件重新分配。

/Project Explorer/工程/Project_Settings/Linker_Files/linker_ram.ld的MEMORY定义：

```
MEMORY
{
  int_itcm              : ORIGIN = 0x00000000, LENGTH = 0x00000000 /* 0KB - Not Supported */
  int_dtcm              : ORIGIN = 0x20000000, LENGTH = 0x00010000 /* 64K */
  int_sram_shareable    : ORIGIN = 0x22C00000, LENGTH = 0x00004000 /* 16KB */
  int_sram              : ORIGIN = 0x34000000, LENGTH = 0x00400000 /* 4MB */
  int_sram_stack_c0     : ORIGIN = 0x34400000, LENGTH = 0x00002000 /* 8KB  */
  int_sram_stack_c1     : ORIGIN = 0x34402000, LENGTH = 0x00002000 /* 8KB  */
  int_sram_stack_c2     : ORIGIN = 0x34404000, LENGTH = 0x00002000 /* 8KB  */
  int_sram_no_cacheable  : ORIGIN = 0x34500000, LENGTH = 0x00100000 /* 1MB, needs to include int_results  */
```

其中，int_sram_no_cacheable被链接到了5M开始地址，而中断向量表intc_vector：

```
.non_cacheable :
  {
    . = ALIGN(4);
    KEEP(*(.int_results))
    . += 0x100;
    . = ALIGN(4096);
    __interrupts_ram_start = .;
    KEEP(*(.intc_vector))
    . = ALIGN(4);
```

就处于这个位置，导致编译出来的binary镜像太大。

vector table在文件/Project Explorer/工程、Project_Settings/Startup_Code/Vector_Table.s中定义的，其属于.intc_vector段，

```
.section  ".intc_vector","ax"
.align 2
.thumb
.globl undefined_handler
.globl undefined_handler
.globl VTABLE
…
```

所以为了减小代码大小，和避开uboot的代码段，如下：

- 修改linker_ram.ld的non_cacheable段，把这部分都放在int_sram，这样编译后你就会发现整个代码就变小了。

```
  .non_cacheable :
  {…
  } > int_sram
```

编译结束后，bin文件变成了257KB。

- 为了避开uboot的代码段，防止互相冲突，我们把M7镜像移动到SRAM的5M位置：

```
    reserveda_to_a53_sram   : ORIGIN = 0x34000000, LENGTH = 0x00500000 /* 5MB */
    int_sram              : ORIGIN = 0x34500000, LENGTH = 0x00080000 /* 512KBsize 5M offsite */
    int_sram_stack_c0     : ORIGIN = 0x34580000, LENGTH = 0x00002000 /* 8KB  */
    int_sram_stack_c1     : ORIGIN = 0x34582000, LENGTH = 0x00002000 /* 8KB  */
    int_sram_stack_c2     : ORIGIN = 0x34584000, LENGTH = 0x00002000 /* 8KB  */
    int_sram_no_cacheable   : ORIGIN = 0x34588000, LENGTH = 0x00078000 /* 480KB, needs to include
int_results  */
    ram_rsvd2             : ORIGIN = 0x34600000, LENGTH = 0         /* End of SRAM 6M offsite */
/*
  * int_sram              : ORIGIN = 0x34000000, LENGTH = 0x00400000 // 4MB
  * int_sram_stack_c0     : ORIGIN = 0x34400000, LENGTH = 0x00002000 // 8KB
  * int_sram_stack_c1     : ORIGIN = 0x34402000, LENGTH = 0x00002000 //8KB
  * int_sram_stack_c2     : ORIGIN = 0x34404000, LENGTH = 0x00002000 // 8KB
  * int_sram_no_cacheable   : ORIGIN = 0x34500000, LENGTH = 0x00100000 // 1MB, needs to include
int_results
  * ram_rsvd2             : ORIGIN = 0x34600000, LENGTH = 0         // End of SRAM
*/
```

- 如果想通过Serial Boot模式启动，就需要在代码中增加对code以外的SRAM初始化。在链接文件中，code的最后增加一个symbol，名为 ”__customer_code_end"，并且定义宏CUSTOMER_CODE_END。同样你也可以定义CUSTOMER_CODE_START。

```
SECTIONS
{

  .sram :
  {
    __customer_code_start = .;
    . = ALIGN(4);
    KEEP(*(.core_loop))
…
 } > int_sram
 .non_cacheable :
  {…
    KEEP(*(.pfe_bd_mem))
    __customer_code_end = .;
  } > int_sram
…
  __INT_SRAM_START       = ORIGIN(reserveda_to_a53_sram); //此为0~5M的SRAM空间，预留给uboot代码
  __INT_SRAM_END         = ORIGIN(int_sram);

 __INT_SRAM_START_2       = __customer_code_end; //此为M7从5M开始后，镜像剩下的空间
  __INT_SRAM_END_2        = ORIGIN(ram_rsvd2);
/*
 *  __INT_SRAM_START       = ORIGIN(int_sram);
 *  __INT_SRAM_END         = ORIGIN(ram_rsvd2);
*/
```

- 使能RAM_INIT

```
    __RAM_INIT            = 1;
```

- 打开文件/Project Explorer/工程/Project_Settings/Startup_Code/startup_cm7.s

默认RamInit是不执行的，因为RAM_INIT默认是0。我们则是利用这个汇编函数，做SRAM ECC的初始化，地址范围是CODE以外的区域，所以是两个区域，需要修改函数

```
RamInit:
  /* Initialize SRAM ECC */
  ldr  r0, =__RAM_INIT
  cmp  r0, 0
  /* Skip if    SRAM  INIT is not set */
  beq SRAM  LOOP  END
  ldr r1, =__INT_SRAM_START
  ldr r2, =__INT_SRAM_END

  subs    r2, r1
  subs    r2, #1
  ble SRAM_LOOP_END

  movs    r0, 0
  movs    r3, 0
SRAM_LOOP:
  stm r1!, {r0,r3}
  subs r2, 8
  bge SRAM_LOOP


  /*johnli remove segment2*/
  ldr r1, =__INT_SRAM_START_2
  ldr r2, =__INT_SRAM_END_2

  subs    r2, r1
  subs    r2, #1
  ble SRAM_LOOP_END

  movs    r0, 0
  movs    r3, 0
SRAM_LOOP_2:
  stm r1!, {r0,r3}
  subs r2, 8
  bge SRAM_LOOP_2

   /*johnli remove segment2 end*/
SRAM_LOOP_END:
```

所以原来的代码是不能执行的，会把整个RAM清掉(传统MCU的代码是在内部Flash中，SRAM是可以全清掉了)。

- 这样操作后，再次编译，bin文件为257KB，和编译后console显示的text的大小是一样的。

```
arm-none-eabi-size --format=berkeley Linflexd_Uart_Ip_Example_S32G274A_M7.elf
   text    data     bss     dec     hex filename
 262152       0   12288  274440   43008 Linflexd_Uart_Ip_Example_S32G274A_M7.elf
```

对应看 map 文件：

```
.sram          0x34500000     0x5c5c
               0x34500000                __customer_code_start = .
…
*(.pfe_bd_mem)
               0x34540000                __customer_code_end = .
```

6.9 M7 的 MPU 设置

MPU 概念：MPU 保护域（ProtectionRegions）ARM 处理器中的 MPU 使用“域（regions）”来对内存单元进行管理。域是与存储空间相关联的属性，处理器核将这些数据保存在协处理器 CP15 的一些寄存器中。通常域的个数为 8 个，编号为从 0～7。域的大小和起始地址保存在 CP15 的寄存器 c6 中。大小可以是 4KB～4GB 的任何 2 的乘幂。域的起始地址必须是其大小的倍数。比如，一个定义为 4KB 的域其起始地址可以是 0x12345000，而一个大小定义为 8KB 的域起始地址只能是 0x2000 的倍数。另外，操作系统可以为这些域分配更多的属性：访问权限、cache 和写缓存。存储器基于当时的处理器模式（管理模式或用户模式）可以设定这些区域的访问权限为读/写、只读和不可访问。

M7 工程源代码中的 MPU 寄存器设定如下：
C:/NXP/S32DS.3.4/S32DS/software/PlatformSDK_S32XX_2021_05/SW32_RTD…/Platform_TS_.../startup/include/core_specific.h

```
/*
  Region  Description       Start      End        Size[KB]  Type          Inner Cache Policy    Outer Cache Policy    Shareable
Executable    Privileged Access    Unprivileged Access
--------  ------------------  ----------  ----------  ----------  ----------------  --------------------  --------------------  -----------
------------  -------------------  ----------------------
…
      5  RAM(1st 4MB)      0x34000000  0x343FFFFF      4096  Normal        Write-Back/Allocate   Write-Back/Allocate
No          Yes          Read/Write          Read/Write
      6  RAM(2MB)          0x34400000  0x345FFFFF      2048  Normal        Write-Back/Allocate   Write-Back/Allocate
No          Yes          Read/Write          Read/Write
      7  Non-Cacheable RAM  0x34500000  0x345FFFFF      1024  Normal          None                  None                  Yes
Yes          Read/Write          Read/Write
…
*/
```

```
    static const uint32 rbar[CPU_MPU_MEMORY_COUNT] = {…, 0x34000000UL, 0x34400000UL,
0x34500000UL, …};
static const uint32 rasr[CPU_MPU_MEMORY_COUNT] = {…, 0x030B002BUL, 0x030B0029UL,
0x130C0027UL, …};
```

rasr 的寄存器定义如下(源之文档《ARM® Cortex®-M7 Processor User Guide Revision: r1p1 Reference Material》)：

The bit assignments are:

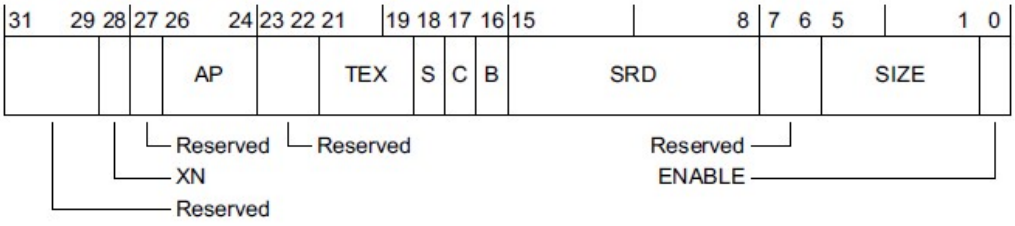


Table 4-53 MPU_RASR bit assignments

Bits	Name	Function
[31:29]	-	Reserved.
[28]	XN	Instruction access disable bit: 0 Instruction fetches enabled. 1 Instruction fetches disabled.
[27]	-	Reserved.
[26:24]	AP	Access permission field, see Table 4-57 on page 4-50.
[23:22]	-	Reserved.
[21:19, 17, 16]	TEX, C, B	Memory access attributes, see Table 4-55 on page 4-49.
[18]	S	Shareable bit, see Table 4-55 on page 4-49.
[15:8]	SRD	Subregion disable bits. For each bit in this field: 0 Corresponding sub-region is enabled. 1 Corresponding sub-region is disabled. See *Subregions* on page 4-52 for more information. Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, write the SRD field as `0x00`.
[7:6]	-	Reserved.
[5:1]	SIZE	Specifies the size of the MPU protection region. The minimum permitted value is 4 (`0b00100`), see *SIZE field values* for more information.
[0]	ENABLE	Region enable bit.

TEX	C	B	S	Memory type	Shareability	Other attributes
0b000	0	0	x[a]	Strongly Ordered	Shareable	-
		1	x[a]	Device	Shareable	-
	1	0	0	Normal	Not shareable	Outer and inner write-through. No Write-Allocate.
			1		Shareable	
		1	0	Normal	Not shareable	Outer and inner Write-Back. No Write-Allocate.
			1		Shareable	
0b001	0	0	0	Normal	Not shareable	Outer and inner noncacheable.
			1		Shareable	
		1	x[a]	Reserved encoding		-
	1	0	x[a]	Implementation defined attributes		-
		1	0	Normal	Not shareable	Outer and inner Write-Back. Write and read allocate.
			1		Shareable	
0b010	0	0	x[a]	Device	Not shareable	Nonshared Device.
		1	x[a]	Reserved encoding		-
	1	x[a]	x[a]	Reserved encoding		-
0b1BB	A	A	0	Normal	Not shareable	Cached memory, BB = outer policy, AA = inner policy. See Table 4-56 on page 4-50 for the encoding of the AA and BB bits.
			1		Shareable	

所以修改了内存分配后，会有两个问题：

- 原来默认的分配在 0x34500000UL 的地址空间是 XN=1 的，也就是说不能执行代码，因为我们把 M7 的代码重新 allocate 到了这个地址，所以 XN 需要修改成 0：0x030C0027UL
- 原来 M7 所在代码的地址空间 0x34000000UL(rasr=0x030B002BUL)的 TEXT C B S 段是：

  001 1 1 0 的，也就是 Normal memory, Not shareable, Outer and inner write-back, write and read allocate 的属性，实际测试中发现打开 cache 后，串口写入此内存的数据在内存中会有不正确的情况，导致 Uboot 执行失败，所以将 cache 关闭，设置为 001 0 0 1 也就是 Normal memory,Shareable, outer and inner noncacheable 的方式，0x34000000UL(rasr=0x030C002BUL)

  当然还有一种更简单的方式就是不设置 MPU 寄存器。

7 测试方法与结果

7.1 下载 M7 镜像

通过 Project Explorer/工程/Project_Settings/Linker_Files/Linker_ram.ld 可以获得 M7 镜像的下载地址

```
MEMORY
{ …
  int_sram              : ORIGIN = 0x34500000, LENGTH = 0x00400000 /* 4MB */
```

通过 Project Explorer/工程/Debug_RAM/工程.map 文件可以获得 M7 镜像启动地址：

```
.intc_vector   0x34506000      0x408 ./Project_Settings/Startup_Code/Vector_Table.o
               0x34506000              VTABLE
```

所以利用 Serial Boot 下载 M7 镜像(修改为 115200_bootloader.bin，放在和 s32gSerialBoot.exe 同一目录，打开 windows 终端)到 S32G 中并执行的命令是：

```
s32gSerialBoot.exe connect COM22 48000 115200_bootloader.bin 0x34500000 0x34506000
```

执行结果：

```
********s32g Serial Boot Loader Tool********
Connecting and overriding intial bootloader
…
Total bytes transferred: 262152
```

7.2 工具的 PING 与帮助功能

下载了 M7 镜像后，执行：

```
s32gSerialBoot.exe ping COM22 115200
********s32g Serial Boot Loader Tool********
PING: The target is running the overriden bootloader
```

可以查看 M7 代码是否运行，帮助说明如下：

```
s32gSerialBoot.exe help
********s32g Serial Boot Loader Tool********
Utility to load a program serially on the s32g

USAGE:
To connect and override the serial boot rom code with m7 kernel bootloader use the command connect
```

Syntax: s32g_serialboot connect COM_PORT baudrate file_name download_address run_address

Example:s32gSerialBoot.exe connect COM22 48000 115200_bootloader.bin 0x34500000 0x34506000

To load a A kernel bootloader to a specific address use the command load

Syntax: s32g_serialboot load COM_PORT baudrate connect download_address

Example: s32gSerialBoot.exe load COM22 115200 u-boot.bin 0x34090000

To start execution, there is the option to start with the M7 or A53 core

Syntax: s32g_serialboot run COM_PORT baudrate kernal_flag execute address

Example: s32gSerialBoot.exe run COM22 115200 A53 0x340a0000

To ping the status of the targed, use the ping command

Syntax: s32g_serialboot ping COM_PORT baudrate

Example: s32g_serialboot ping COM22 115200

7.3 下载 Uboot 镜像

根据文档《S32G_Uboot_BSPXX_VX-XXXXXXXX.pdf》创建 Uboot 的 standalone 编译环境。

根据文件：/u-boot/board/freescale/s32-gen1/Kconfig/可以得到

```
config SYS_TEXT_BASE
	default 0xff8a0000 if S32_ATF_BOOT_FLOW
	default 0x340a0000
```

在 make s32g274ardb2_defconfig 后，也可以从/u-boot/.config 中看到：

```
CONFIG_SYS_TEXT_BASE=0x340a0000
```

在编译结束后，从/u-boot/u-boot.map 中也可以得到此执行地址：

```
Linker script and memory map
Address of section .text set to 0x340a0000
                0x0000000000000000                . = 0x0
                0x0000000000000000                . = ALIGN (0x8)

.text           0x00000000340a0000    0x607dc
 *(.__image_copy_start)
 .__image_copy_start
```

```
                0x00000000340a0000        0x0 arch/arm/lib/built-in.o
                0x00000000340a0000                __image_copy_start
 arch/arm/cpu/armv8/start.o(.text*)
 .text          0x00000000340a0000      0x1e0 arch/arm/cpu/armv8/start.o
                0x00000000340a0000                _start
```

但是要注意一下：

Arch\arm\mach-s32\makefile

```
u-boot.s32: u-boot.bin FORCE
ifeq ($(CONFIG_S32_GEN1),y)
  $(eval DTB_RSRVD_SIZE = $(shell \
                echo $$(($(CONFIG_SYS_TEXT_BASE) - $(CONFIG_DTB_SRAM_ADDR)))))
  $(eval DTB_SIZE = $(shell stat --printf="%s" u-boot.dtb))

  @if [ $(DTB_SIZE) -gt $(DTB_RSRVD_SIZE) ]; then \
                echo "DTB exceeds the reserved space of" \
                "$(DTB_RSRVD_SIZE) bytes between CONFIG_SYS_TEXT_BASE" \
                "and CONFIG_DTB_SRAM_ADDR"; \
                false; \
  fi

  @dd if=u-boot.dtb of=u-boot-with-dtb.bin bs=$(DTB_RSRVD_SIZE) count=1
  @dd if=u-boot.bin of=u-boot-with-dtb.bin bs=$(DTB_RSRVD_SIZE) seek=1
```

所以 Makefile 会把 Uboot 的 DTB Image 打包到 Uboot 的头前面，从 CONFIG_DTB_SRAM_ADDR 开始。

宏定义在/u-boot/board/freescale/s32-gen1/Kconfig 中：

```
config DTB_SRAM_ADDR
  hex "SRAM address at which the dtb will be found" if !S32_ATF_BOOT_FLOW
  default 0xff890000 if S32_ATF_BOOT_FLOW
  default 0x34090000
```

所以 Uboot 的下载地址是 0x34090000,执行地址是 0x340a0000：Uboot 镜像下载执行命令是：

```
s32gSerialBoot.exe load COM22 115200 u-boot.bin 0x34090000
```

注意要下载*.bin 不是*.s32 文件。

执行结果：

```
********s32g Serial Boot Loader Tool********
Loading binary file

write data length 785375
bytesTransfered pkt size 785375, checksum ok
Transfer complete
```

7.4 执行启动 A53 命令

根据以上分析， 让 M7 去启动 A53 的启动地址是 0x340a0000，所以执行命令是：

```
s32gSerialBoot.exe run COM22 115200 A53 0x340a0000
```

执行结果在 PC 端：

```
********s32g Serial Boot Loader Tool********
Launching exection command
execute checksum ok
```

在 PC 端，等 s32gSerialBoot.exe 执行结束后，打开串口，配置为 115200,8bit,1bit 停止位，无奇偶校验，无停止位，可以看到 Uboot 已经正确执行。

7.5 综合测试

如 5.5 与 6.7 所说，代码实现了下载与启动一体化功能，并提供是否打开 checksum 的能力，所以实现一个 bat 文件如下：

```
s32gSerialBoot.exe connect COM22 48000 115200_bootloader.bin 0x34500000 0x34506000
echo wait for some time
TIMEOUT /T 3
s32gSerialBoot.exe load_to_run COM22 115200 u-boot.bin 0x34090000 0x340A0000 false
```

双击运行结束后打开串口，可以看到 Uboot 已经执行。

7.6 速率极限测试

在 M7 镜像中，将串口波特率改成最高：
ConfigTools->Peripherals->Components->Linflexd_Uart_1->UartGlobalConfig:

将 Desire Baudrate 改成：LINFLEXD_UART_BAUDRATE_921600。

修改结束后 Update Code，查看源代码如下：generate/src/Linflexd_Uart_lp_VS_0_PBcfg.c

```
const Linflexd_Uart_Ip_UserConfigType Linflexd_Uart_Ip_pHwConfigPB_0_VS_0 =
{
  /* Actual baudrate */
  921600,
```

然后编译生成的 bin 文件拷贝为 921600_bootloader.bin 并执行，

```
s32gSerialBoot.exe connect COM22 48000 921600_bootloader.bin 0x34500000 0x34506000
********s32g Serial Boot Loader Tool********
Connecting and overriding intial bootloader
Total bytes transferred: 4096
…
Total bytes transferred: 262152
s32gSerialBoot.exe load_to_run COM22 921600 u-boot.bin 0x34090000 0x340A0000 true
********s32g Serial Boot Loader Tool********
Launching load to run command
header sum=25598
write data length 785375
sum=4191,csum=-4192
bytesTransfered pkt size 785375, checksum ok
Transfer complete:785375
```

然后打开串口，Uboot 正常执行。

而 baudrate 设置为 LINFLEXD_UART_BAUDRATE_1843200,则测试失败，可见目前的实现方法，在 S32G RDB2 板上可以测试到 1M 左右的波特率。

8 发布说明

本工具发布的代码包目录说明如下：

Release

|->M7: Linflexd_Uart_Ip_Example_S32G274A_M7: S32DS M7 工程。

|->PC: s32gSerialBoot_Csharp: PC 端的 Visual Studio 的 C#的串口工具工程。

|->Test:

| |-> 115200_bootloader.bin: S32DS M7 工程编译出来的 bin 文件，波特率为 115200

| |-> 921600_bootloader.bin: S32DS M7 工程编译出来的 bin 文件，波特率为 921600

| |->load_uboot.bat: 运行工具的批处理文件，运行成功后打开串口可以看到 Uboot 执行，默认使用的波特率是 115299

| |->readme.txt:其它测试命令

| |->s32gSerialBoot.exe:编译出来的 PC 端串口工具

| |->u-boot.bin: BSP29 默认编译出来的 u-boot.bin.

9 二次开发说明

本工具及文档是展示如何通过serial boot功能运行一个uboot程序，这个是作为研发或工厂on-line烧写镜像的一个基础，运行了uboot后，就有机会操作一个高速接口(如以太网或USB接口)来烧写内核，并且uboot+Linux内核也可以提供更多，更灵活的烧写工具，而on-line烧写的办法依赖与客户定制，没有一定之规，大概的思路有：

- 通过Uboot的以太网驱动，tftp入内核镜像，然后mount网终文件系统，来运行一个Linux来进行烧录，或者tftp内核+ramrootfs也可以。这个需要定制Uboot的以太网PHY或者Switcher驱动，并定制相关Linux工具。
- 通过Uboot使用USB fastboot来加载Linux kernel+ramfs镜像，不过由于S32G2没有集成USB PHY，需要外接，而且汽车网关把USB设计出来的也比较少，所以并不适用。

由于在线烧写的具体实现属于客户定制，本文不在展开讨论，本文的目的是实现serial boot启动A核，解决其中的关键技术问题，从而提供给客户一个基础。

至于工具本身，为了适应目前的实现办法，所以是使用同步非中断1 Byte的方式使用串口的，如果需要提高效率，可以考虑改成异步中断多Bytes读写，但是需要适配实现。