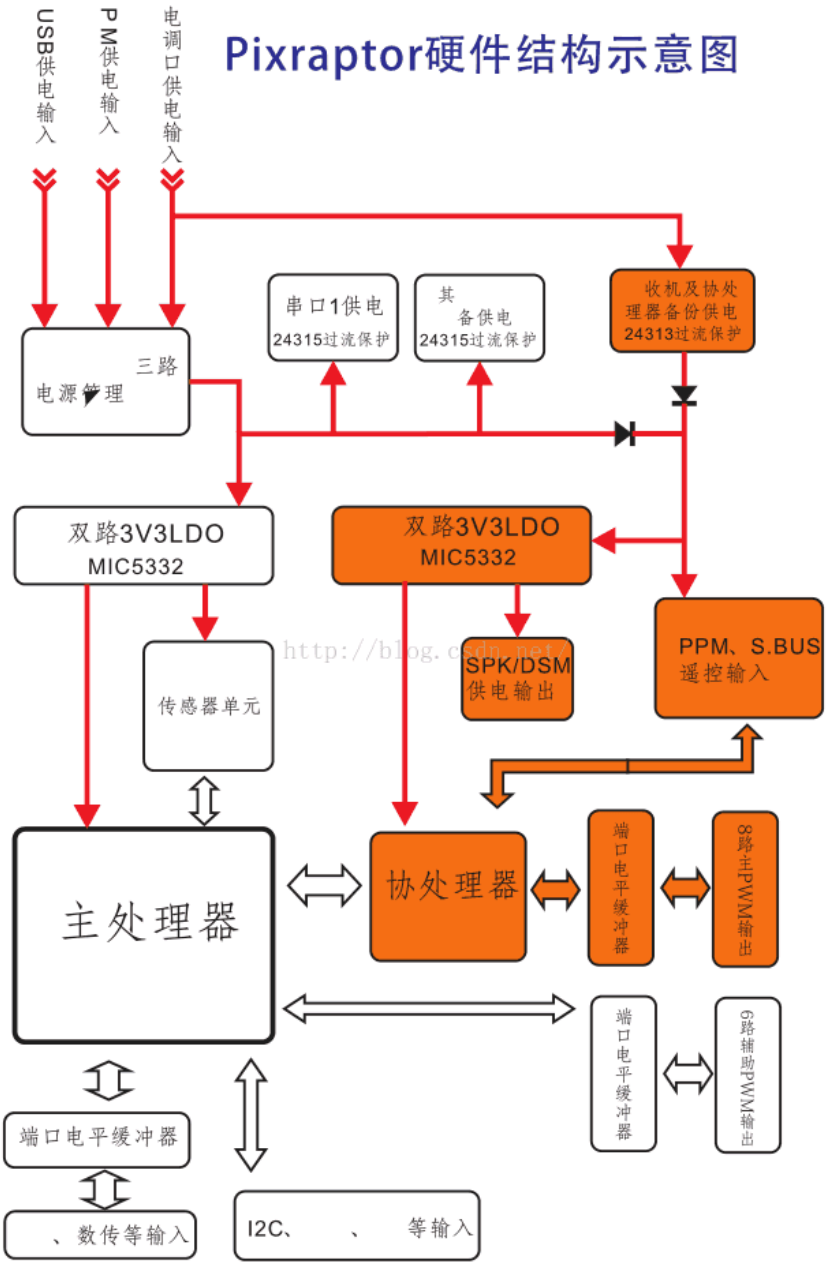
PIXHAWK PX4FMU和PX4IO最底层启动过程分析

首先，大体了解PX4IO 与PX4FMU各自的任务

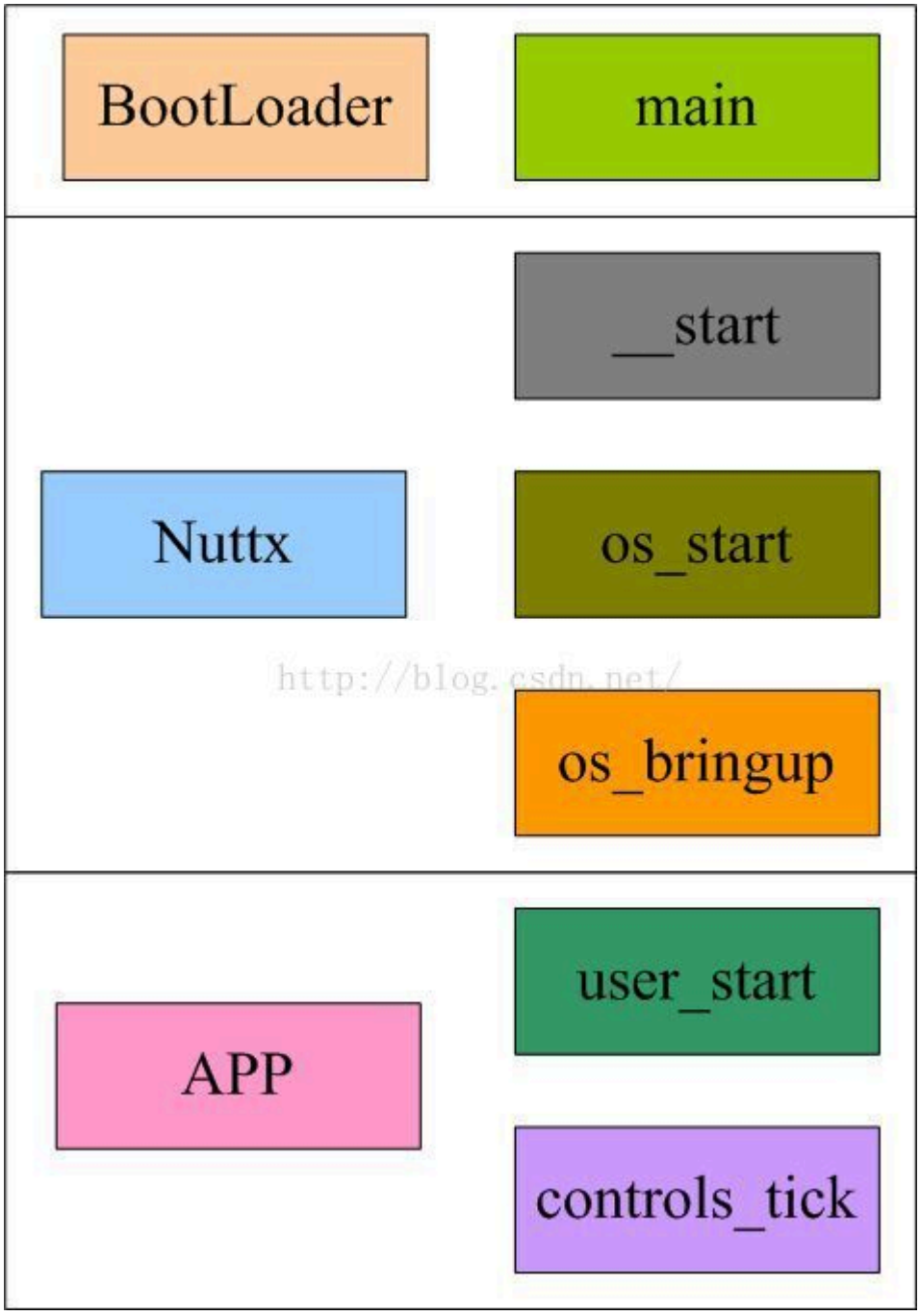


PX4IO(STM32F100)为PIXHAWK 中专用于处理输入输出的部分,输入为支持的各类遥控器(PPM,SPKT/DSM,SBUS), 输出为电调的PWM 驱动信号, 它与PX4FMU(STM32F427)通过串口进行通信

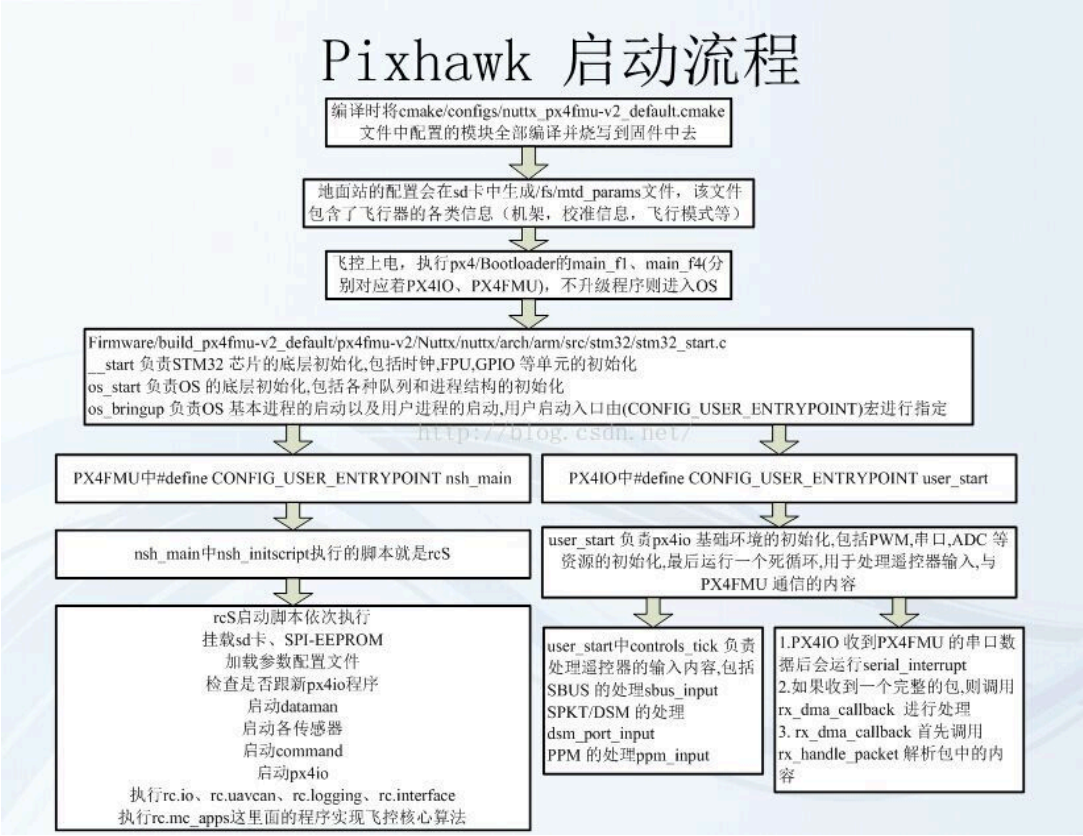
PX4FMU： 各种传感器数据读取、姿态解算、PWM控制量的计算、与PX4IO通信

进入主题

PX4IO 与PX4FMU 一样,在大的软件架构上分为Bootloader,OS 和APP 三个部分,OS 为Nuttx,如下图



根据自己理解画的流程图： （2016.05.19加）



1.Bootloader

在嵌入式操作系统中，BootLoader是在操作系统内核运行之前运行。可以初始化硬件设备、建立内存空间映射图，从而将系统的软硬件环境带到一个合适状态，以便为最终调用操作系统内核准备好正确的环境。在嵌入式系统中，通常并没有像BIOS那样的固件程序（注，有的嵌入式CPU也会内嵌一段短小的启动程序），因此整个系统的加载启动任务就完全由BootLoader来完成。Bootloader是嵌入式系统在加电后执行的第一段代码，在它完成CPU和相关硬件的初始化之后，再将操作系统映像或固化的嵌入式应用程序装在到内存中然后跳转到操作系统所在的空间，启动操作系统运行。

因此进入px4/Bootloader可以看到main_f1、main_f4，分别对应着PX4IO、PX4FMU

主要是负责是否升级芯片,不升级则进入OS（具体的内容还没仔细看，读者结合注释应该能够看懂）

2.Nuttx

此部分摘自于pixhawk自学笔记之px4程序启动顺序

代码位置：Firmware/build_px4fmu-v2_default/px4fmu-v2/Nuttx/nuttx/arch/arm/src/stm32/stm32_start.c

__start-- #处理器执行的第一条指令（px4使用的是stm32，入口在stm32_start.c中）

|

v

stm32_clockconfig()------ #初始化时钟

|

v

rcc_reset() #复位rcc

stm32_stdclockconfig() #初始化标准时钟

rcc_enableperipherals() #使能外设时钟

---

stm32_fpuconfig() #配置fpu

|

v

stm32_lowsetup() #基本初始化串口，之后可以使用up_lowputc()

stm32_gpioinit() #初始化gpio，只是调用stm32_gpioremap()设置重映射

up_earlyserialinit() #初始化串口，之后可以使用up_putc()

stm32_boardinitialize()-- #板级初始化

|

v

stm32_spiinitialize() #初始化spi，只是调用stm32_configgpio()设置gpio

stm32_usbinitialize()　　#初始化usb，只是调用stm32_configgpio()设置gpio

up_ledinit();　　#初始化led，只是调用stm32_configgpio()设置gpio

|

在stm32_start.c文件中我们会看到这么一句话：

```
/* Then start NuttX */

showprogress('\r');
showprogress('\n');
os_start();
```


os_start()----------　　#初始化操作系统

|

v

dq_init()　　#初始化各种状态的任务列表（置为null）

g_pidhash[i]=　　#初始化唯一可以确定的元素--进程ID

g_pidhash[PIDHASH(0)]=　　#分配空闲任务的进程ID为0

g_idletcb=　　#初始化空闲任务的任务控制块

sem_initialize()--　　#初始化信号量

|

v

dq_init()　　#将信号量队列置为null

sem_initholders()　　#初始化持有者结构以支持优先级继承

|

|

v

up_allocate_heap()　　#分配用户模式的堆（设置堆的起点和大小）

kumm_initialize()　　#初始化用户模式的堆

up_allocate_kheap()　　#分配内核模式的堆

kmm_initialize()　　#初始化内核模式的堆

task_initialize()　　#初始化任务数据结构

irq_initialize()　　#将所有中断向量都指向同一个异常中断处理程序

wd_initialize()　　#初始化看门狗数据结构

clock_initialize()　　#初始化rtc

timer_initialize()　　#配置POSIX定时器

sig_initialize()　　#初始化信号

mq_initialize()　　#初始化命名消息队列

pthread_initialize()　　#初始化线程特定的数据，空函数

fs_initialize()---　　#初始化文件系统

|

v

sem_init()　　#初始化节点信号量为1

files_initialize() #初始化文件数组，空函数

|

|

v

net_initialize()--　　#初始化网络

　　|

　　v

uip_initialize()　　#初始化uIP层

net_initroute()　　#初始化路由表

netdev_seminit()　#初始化网络设备信号量

arptimer_init()　　#初始化ARP定时器

　　|

|

v

up_initialize()---　　#处理器特定的初始化

　　|

　　v

up_calibratedelay()　　#校准定时器

up_addregion()　　#增加额外的内存段

up_irqinitialize()　　#设置中断优先级，关联硬件异常处理函数

up_pminitialize()　　#初始化电源管理

up_dmainitialize()　　#初始化DMA

up_timerinit()　　#初始化定时器中断

devnull_register()　　#注册/dev/null

devzero_register()　　#注册/dev/zero

up_serialinit()　　#注册串口控制台/dev/console和串口/dev/ttyS0

up_rnginitialize()　　#初始化并注册随机数生成器

up_netinitialize()　　#初始化网络，是arch/arm/src/chip/stm32_eth.c中的

up_usbinitialize()　　#初始化usb驱动

board_led_on()　　#打开中断使能led，但很快会被其它地方的led操作改变状态

|

|

v

lib_initialize()　　#初始化c库，空函数

group_allocate()　　#分配空闲组

group_setupidlefiles()　　#在空闲任务上创建stdout、stderr、stdin

group_initialize()　　#完全初始化空闲组

os_bringup()------　　#创建初始任务

|

v

KEKERNEL_THREAD()　#启动内核工作者线程

board_initialize()　　#最后一刻的板级初始化

TASK_CREATE()　　#启动默认应用程序

|

|

v

forup_idle()　　#空闲任务循环

|

|

v

for(;;)　　#不应该到达这里

__start 负责STM32 芯片的底层初始化,包括时钟,FPU,GPIO 等单元的初始化

os_start 负责OS 的底层初始化,包括各种队列和进程结构的初始化

os_bringup 负责OS 基本进程的启动以及用户进程的启动,用户启动入口由(CONFIG_USER_ENTRYPOINT)宏进行指定

3.APP

PX4FMU中　#define CONFIG_USER_ENTRYPOINT nsh_main

进入nsh_main

```
1. int nsh_main(int argc, char *argv[])
2. {
3. int exitval = 0;
4. int ret;
5.
6. /* Call all C++ static constructors */
7.
8. #if defined(CONFIG_HAVE_CXX) && defined(CONFIG_HAVE_CXXINITIALIZE)
9. up_cxxinitialize();
10. #endif
```

```
11.
12.  /* Make sure that we are using our symbol take */
13.
14.  #if defined(CONFIG_LIBC_EXECFUNCS) && defined(CONFIG_EXECFUNCS_SYMTAB)
15.  exec_setsymtab(CONFIG_EXECFUNCS_SYMTAB, 0);
16.  #endif
17.
18.  /* Register the BINFS file system */
19.
20.  #if defined(CONFIG_FS_BINFS) && (CONFIG_BUILTIN)
21.  ret = builtin_initialize();
22.  if (ret < 0)
23.  {
24.  fprintf(stderr, "ERROR: builtin_initialize failed: %d\n", ret);
25.  exitval = 1;
26.  }
27.  #endif
28.
29.  /* Initialize the NSH library */
30.
31.  nsh_initialize();
32.
```

```
/* If the Telnet console is selected as a front-end, then start the
 * Telnet daemon.
 */

#ifdef CONFIG_NSH_TELNET
ret = nsh_telnetstart();
if (ret < 0)
{
  /* The daemon is NOT running. Report the the error then fail...
   * either with the serial console up or just exiting.
   */

  fprintf(stderr, "ERROR: Failed to start TELNET daemon: %d\n", ret);
  exitval = 1;
}
#endif

/* If the serial console front end is selected, then run it on this thread */

#ifdef CONFIG_NSH_CONSOLE
```

```
53.  ret = nsh_consolemain(0, NULL);
54.
55.  /* nsh_consolemain() should not return. So if we get here, something
56.  * is wrong.
57.  */
58.
59.  fprintf(stderr, "ERROR: nsh_consolemain() returned: %d\n", ret);
60.  exitval = 1;
61.  #endif
62.
63.  return exitval;
64.  }
```

其中包含

```
1.  #ifdef CONFIG_NSH_CONSOLE
2.  ret = nsh_consolemain(0, NULL);
```

进入nsh_consolemain

```
int nsh_consolemain(int argc, char *argv[])
{
FAR struct console_stdio_s *pstate = nsh_newconsole();
int ret;

DEBUGASSERT(pstate);

/* Execute the start-up script */

#ifdef CONFIG_NSH_ROMFSETC
(void)nsh_initscript(&pstate->cn_vtbl);
#endif

/* Initialize any USB tracing options that were requested */

#ifdef CONFIG_NSH_USBDEV_TRACE
usbtrace_enable(TRACE_BITSET);
#endif

/* Execute the session */

```

```
22.  ret = nsh_session(pstate);
23.
24.  /* Exit upon return */
25.
26.  nsh_exit(&pstate->cn_vtbl, ret);
27.  return ret;
28.  }
```

其中包含

```
1.  /* Execute the start-up script */
2.
3.  #ifdef CONFIG_NSH_ROMFSETC
4.  (void)nsh_initscript(&pstate->cn_vtbl);
5.  #endif
```

执行启动脚本也就是rcS，接下来根据自己版本分别看ardupilot和PX4原生码

```
1.  /* Execute the session */
2.
3.  ret = nsh_session(pstate);
```

执行用户程序
跟踪pstate

```
FAR struct console_stdio_s *pstate = nsh_newconsole();
```

进入nsh_newconsole

```
1.  FAR struct console_stdio_s *nsh_newconsole(void)
2.  {
3.  struct console_stdio_s *pstate = (struct console_stdio_s *)zalloc(sizeof(struct console_stdio_s));
4.  if (pstate)
5.  {
6.  /* Initialize the call table */
7.
8.  #ifndef CONFIG_NSH_DISABLEBG
9.  pstate->cn_vtbl.clone = nsh_consoleclone;
10. pstate->cn_vtbl.release = nsh_consolerelease;
11. #endif
12. pstate->cn_vtbl.write = nsh_consolewrite;
13. pstate->cn_vtbl.output = nsh_consoleoutput;
14. pstate->cn_vtbl.linebuffer = nsh_consolelinebuffer;
15. pstate->cn_vtbl.redirect = nsh_consoleredirect;
```

```
16. pstate->cn_vtbl.undirect = nsh_consoleundirect;
17. pstate->cn_vtbl.exit = nsh_consoleexit;
18.
19. /* (Re-) open the console input device */
20.
21. #ifdef CONFIG_NSH_CONDEV
22. pstate->cn_confd = open(CONFIG_NSH_CONDEV, O_RDWR);
23. if (pstate->cn_confd < 0)
24. {
25. free(pstate);
26. return NULL;
27. }
28.
29. /* Create a standard C stream on the console device */
30.
31. pstate->cn_constream = fdopen(pstate->cn_confd, "r+");
32. if (!pstate->cn_constream)
33. {
34. close(pstate->cn_confd);
35. free(pstate);
36. return NULL;
37. }
38. #endif
```

```
39.
40.  /* Initialize the output stream */
41.
42.  pstate->cn_outfd = OUTFD(pstate);
43.  pstate->cn_outstream = OUTSTREAM(pstate);
44.  }
45.  return pstate;
46.  }
```

应该是用户在console输入新的nsh命令吧

PX4IO中　　#define CONFIG_USER_ENTRYPOINT user_start

进入user_start

```
1.  int
2.  user_start(int argc, char *argv[])
3.  {
4.  /* configure the first 8 PWM outputs (i.e. all of them)
    */
5.  up_pwm_servo_init(0xff);
6.
7.  /* run C++ ctors before we go any further */
```

```
8.  up_cxxinitialize();
9.
10.  /* reset all to zero */
11.  memset(&system_state, 0, sizeof(system_state));
12.
13.  /* configure the high-resolution time/callout interface
*/
14.  hrt_init();
15.
16.  /* calculate our fw CRC so FMU can decide if we need to
update */
17.  calculate_fw_crc();
18.
19.  /*
20.  * Poll at 1ms intervals for received bytes that have
not triggered
21.  * a DMA event.
22.  */
23.  #ifdef CONFIG_ARCH_DMA
24.  hrt_call_every(&serial_dma_call,       1000,       1000,
(hrt_callout)stm32_serial_dma_poll, NULL);
25.  #endif
26.
27.  /* print some startup info */
```

```
28. lowsyslog("\nPX4IO: starting\n");
29.
30. /* default all the LEDs to off while we start */
31. LED_AMBER(false);
32. LED_BLUE(false);
33. LED_SAFETY(false);
34. #ifdef GPIO_LED4
35. LED_RING(false);
36. #endif
37.
38. /* turn on servo power (if supported) */
39. #ifdef POWER_SERVO
40. POWER_SERVO(true);
41. #endif
42.
43. /* turn off S.Bus out (if supported) */
44. #ifdef ENABLE_SBUS_OUT
45. ENABLE_SBUS_OUT(false);
46. #endif
47.
48. /* start the safety switch handler */
49. safety_init();
50.
```

```
51. /* initialise the control inputs */
52. controls_init();
53.
54. /* set up the ADC */
55. adc_init();
56.
57. /* start the FMU interface */
58. interface_init();
59.
60. /* add a performance counter for mixing */
61. perf_counter_t mixer_perf = perf_alloc(PC_ELAPSED,
"mix");
62.
63. /* add a performance counter for controls */
64. perf_counter_t controls_perf = perf_alloc(PC_ELAPSED,
"controls");
65.
66. /* and one for measuring the loop rate */
67. perf_counter_t loop_perf = perf_alloc(PC_INTERVAL,
"loop");
68.
69. struct mallinfo minfo = mallinfo();
70. lowsyslog("MEM: free %u, largest %u\n", minfo.mxordblk,
minfo.fordblks);
71.
```

```
/* initialize PWM limit lib */
pwm_limit_init(&pwm_limit);

/*
* P O L I C E L I G H T S
*
* Not enough memory, lock down.
*
* We might need to allocate mixers later, and this will
* ensure that a developer doing a change will notice
* that he just burned the remaining RAM with static
* allocations. We don't want him to be able to
* get past that point. This needs to be clearly
* documented in the dev guide.
*
*/
if (minfo.mxordblk < 600) {

lowsyslog("ERR: not enough MEM");
bool phase = false;

while (true) {
```

```
94.
95.  if (phase) {
96.  LED_AMBER(true);
97.  LED_BLUE(false);
98.
99.  } else {
100.  LED_AMBER(false);
101.  LED_BLUE(true);
102.  }
103.
104.  up_udelay(250000);
105.
106.  phase = !phase;
107.  }
108.  }
109.
110.  /* Start the failsafe led init */
111.  failsafe_led_init();
112.
113.  /*
114.  * Run everything in a tight loop.
115.  */
116.
```

```
117. uint64_t last_debug_time = 0;
118. uint64_t last_heartbeat_time = 0;
119.
120. for (;;) {
121.
122. /* track the rate at which the loop is running */
123. perf_count(loop_perf);
124.
125. /* kick the mixer */
126. perf_begin(mixer_perf);
127. mixer_tick();
128. perf_end(mixer_perf);
129.
130. /* kick the control inputs */
131. perf_begin(controls_perf);
132. controls_tick();
133. perf_end(controls_perf);
134.
135. if ((hrt_absolute_time() - last_heartbeat_time) > 250 *
1000) {
136. last_heartbeat_time = hrt_absolute_time();
137. heartbeat_blink();
138. }
```

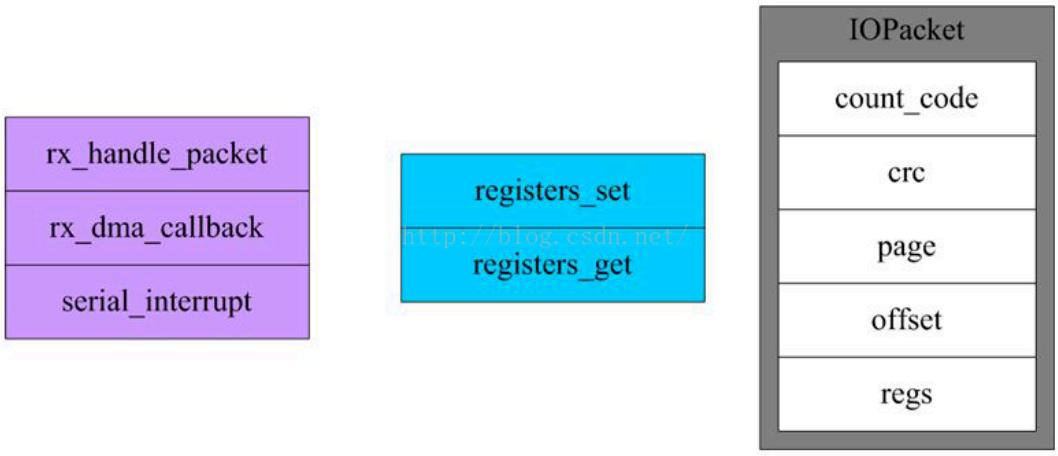
```
139.
140.  ring_blink();
141.
142.  check_reboot();
143.
144.  /* check for debug activity (default: none) */
145.  show_debug_messages();
146.
147.  /* post debug state at ~1Hz - this is via an auxiliary serial port
148.  * DEFAULTS TO OFF!
149.  */
150.  if (hrt_absolute_time() - last_debug_time > (1000 * 1000)) {
151.
152.  isr_debug(1, "d:%u s=0x%x a=0x%x f=0x%x m=%u",
153.  (unsigned)r_page_setup[PX4IO_P_SETUP_SET_DEBUG],
154.  (unsigned)r_status_flags,
155.  (unsigned)r_setup_arming,
156.  (unsigned)r_setup_features,
157.  (unsigned)mallinfo().mxordblk);
158.  last_debug_time = hrt_absolute_time();
159.  }
160.  }
```

```
}
```

user_start 负责px4io 基础环境的初始化,包括PWM,串口,ADC 等资源的初始化,最后运行一个死循环,用于处理遥控器输入,与PX4FMU 通信的内容

controls_tick 负责处理遥控器的输入内容,包括 SBUS 的处理sbus_input、 SPKT/DSM 的处理dsm_port_input、 PPM 的处理ppm_input

PX4IO 底层中断处理的内容如下图



（1）紫色为PX4IO 的底层串口IO 操作,流程为当PX4IO 收到PX4FMU 的串口数据后会运行serial_interrupt, serial_interrupt 负责收发DMA 的操作,如果收到一个完整的包,则调用rx_dma_callback 进行处理, rx_dma_callback 首先调用rx_handle_packet 解析包中的内容,判断为写寄存器还是读寄存器,处理完成后由rx_dma_callback 发送回包给PX4FMU

```
static int
serial_interrupt(int irq, void *context)
{
static bool abort_on_idle = false;

uint32_t sr = rSR; /* get UART status register */
```

```
(void)rDR; /* required to clear any of the interrupt status that brought us here */

if (sr & (USART_SR_ORE | /* overrun error - packet was too big for DMA or DMA was too slow */
USART_SR_NE | /* noise error - we have lost a byte due to noise */
USART_SR_FE)) { /* framing error - start/stop bit lost or line break */

perf_count(pc_errors);

if (sr & USART_SR_ORE) {
perf_count(pc_ore);
}

if (sr & USART_SR_NE) {
perf_count(pc_ne);
}

if (sr & USART_SR_FE) {
perf_count(pc_fe);
}
```

```
/* send a line break - this will abort transmission/reception on the other end */
rCR1 |= USART_CR1_SBK;

/* when the line goes idle, abort rather than look at the packet */
abort_on_idle = true;
}

if (sr & USART_SR_IDLE) {

/*
* If we saw an error, don't bother looking at the packet - it should have
* been aborted by the sender and will definitely be bad. Get the DMA reconfigured
* ready for their retry.
*/
if (abort_on_idle) {

abort_on_idle = false;
dma_reset();
return 0;
}
```

```
/*
 * The sender has stopped sending - this is probably the end of a packet.
 * Check the received length against the length in the header to see if
 * we have something that looks like a packet.
 */
unsigned length = sizeof(dma_packet) - stm32_dmaresidual(rx_dma);

if ((length < 1) || (length < PKT_SIZE(dma_packet))) {

/* it was too short - possibly truncated */
perf_count(pc_badidle);
dma_reset();
return 0;
}

/*
 * Looks like we received a packet. Stop the DMA and go process the
 * packet.
 */
```

```
perf_count(pc_idle);
stm32_dmastop(rx_dma);
rx_dma_callback(rx_dma, DMA_STATUS_TCIF, NULL);
}

return 0;
}
```

```
static void
rx_dma_callback(DMA_HANDLE handle, uint8_t status, void *arg)
{
/*
* We are here because DMA completed, or UART reception stopped and
* we think we have a packet in the buffer.
*/
perf_begin(pc_txns);

/* disable UART DMA */
rCR3 &= ~(USART_CR3_DMAT | USART_CR3_DMAR);

/* handle the received packet */
```

```
14.  rx_handle_packet();
15.
16.  /* re-set DMA for reception first, so we are ready to receive before we start sending */
17.  dma_reset();
18.
19.  /* send the reply to the just-processed request */
20.  dma_packet.crc = 0;
21.  dma_packet.crc = crc_packet(&dma_packet);
22.  stm32_dmasetup(
23.  tx_dma,
24.  (uint32_t)&rDR,
25.  (uint32_t)&dma_packet,
26.  PKT_SIZE(dma_packet),
27.  DMA_CCR_DIR |
28.  DMA_CCR_MINC |
29.  DMA_CCR_PSIZE_8BITS |
30.  DMA_CCR_MSIZE_8BITS);
31.  stm32_dmastart(tx_dma, NULL, NULL, false);
32.  rCR3 |= USART_CR3_DMAT;
33.
34.  perf_end(pc_txns);
35.  }
```

```
static void
rx_handle_packet(void)
{
/* check packet CRC */
uint8_t crc = dma_packet.crc;
dma_packet.crc = 0;

if (crc != crc_packet(&dma_packet)) {
perf_count(pc_crcerr);

/* send a CRC error reply */
dma_packet.count_code = PKT_CODE_CORRUPT;
dma_packet.page = 0xff;
dma_packet.offset = 0xff;

return;
}

if (PKT_CODE(dma_packet) == PKT_CODE_WRITE) {

/* it's a blind write - pass it on */
```

```
22. if   (registers_set(dma_packet.page,   dma_packet.offset,
    &dma_packet.regs[0], PKT_COUNT(dma_packet))) {
23. perf_count(pc_regerr);
24. dma_packet.count_code = PKT_CODE_ERROR;
25.
26. } else {
27. dma_packet.count_code = PKT_CODE_SUCCESS;
28. }
29.
30. return;
31. }
32.
33. if (PKT_CODE(dma_packet) == PKT_CODE_READ) {
34.
35. /* it's a read - get register pointer for reply */
36. unsigned count;
37. uint16_t *registers;
38.
39. if   (registers_get(dma_packet.page,   dma_packet.offset,
    &registers, &count) < 0) {
40. perf_count(pc_regerr);
41. dma_packet.count_code = PKT_CODE_ERROR;
42.
43. } else {
```

```
/* constrain reply to requested size */
if (count > PKT_MAX_REGS) {
count = PKT_MAX_REGS;
}

if (count > PKT_COUNT(dma_packet)) {
count = PKT_COUNT(dma_packet);
}

/* copy reply registers into DMA buffer */
memcpy((void *)&dma_packet.regs[0], registers, count *
2);
dma_packet.count_code = count | PKT_CODE_SUCCESS;
}

return;
}

/* send a bad-packet error reply */
dma_packet.count_code = PKT_CODE_CORRUPT;
dma_packet.page = 0xff;
dma_packet.offset = 0xfe;
}
```

(2) 蓝色为包操作,只提供registers_set 写操作和registers_get 读操作

(3) IOPacket 为协议包,包括以下几部分

count_code 标记包的读写,错误,长度等信息

crc 为包的效验码

page 为数据页

offset 为数据偏移量

regs 为数据内容

数据表格如下:

PAGE	定义	描述
0	PX4IO_PAGE_CONFIG	配置信息,包含协议版本,硬件版本,最大 RC 数量,最大 ADC 数量等信息
1	PX4IO_PAGE_STATUS	状态信息,包含 PPM,SBUS 等通道是否有效等信息
2	PX4IO_PAGE_ACTUATORS	
3	PX4IO_PAGE_SERVOS	
4	PX4IO_PAGE_RAW_RC_INPUT	RC 输入状态,包括信号强度等
5	PX4IO_PAGE_RC_INPUT	RC 输入值
6	PX4IO_PAGE_RAW_ADC_INPUT	ADC 采样值
7	PX4IO_PAGE_PWM_INFO	
50	PX4IO_PAGE_SETUP	
51	PX4IO_PAGE_CONTROLS	
52	PX4IO_PAGE_MIXERLOAD	
53	PX4IO_PAGE_RC_CONFIG	RC 配置信息,例如最大值,最小值,死区等
54	PX4IO_PAGE_DIRECT_PWM	
55	PX4IO_PAGE_FAILSAFE_PWM	
56	PX4IO_PAGE_SENSORS	
57	PX4IO_PAGE_TEST	
106	PX4IO_PAGE_CONTROL_MIN_PWM	
107	PX4IO_PAGE_CONTROL_MAX_PWM	
108	PX4IO_PAGE_DISARMED_PWM	