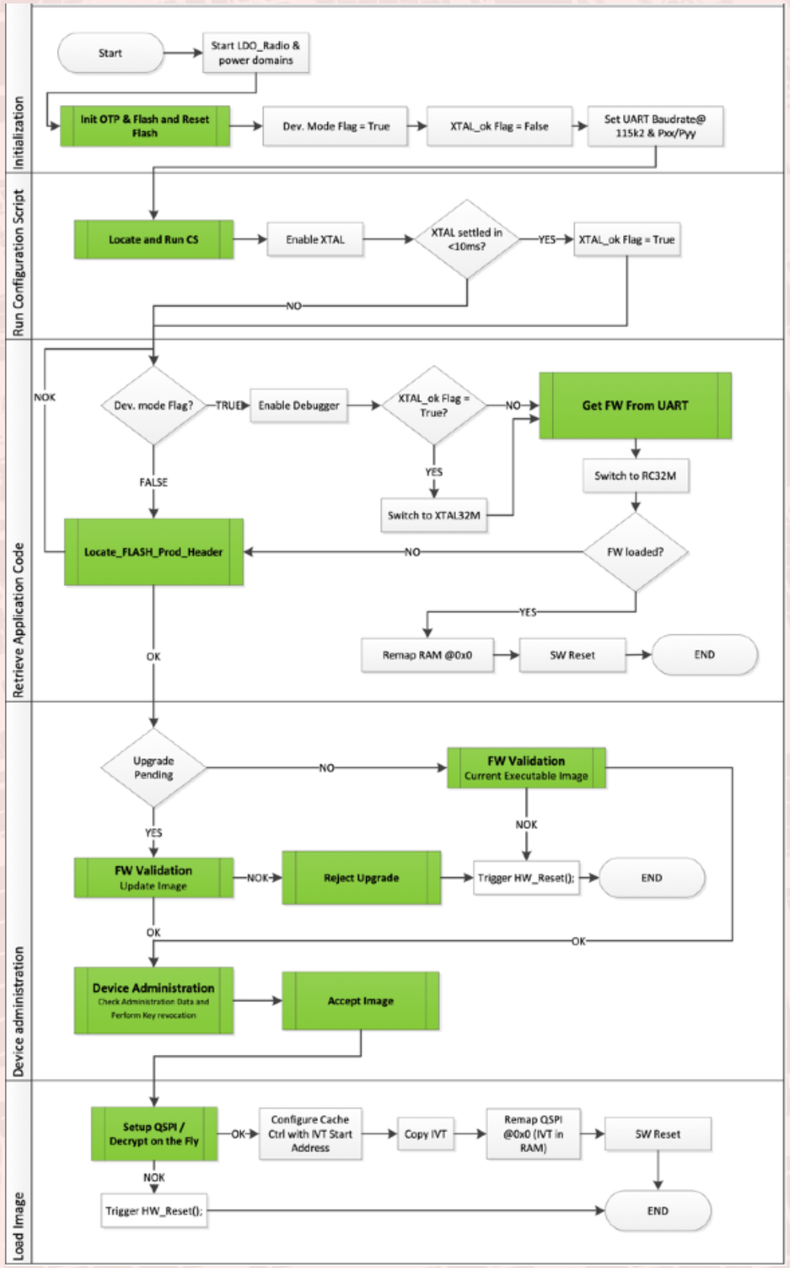


```
CLK_Enable_RC32M();
CRG_TOP_CLK_AMBA_REG[0] = 0;                    // reset peripherals clocking
SYS_WDOG_WATCHDOG_CTRL_REG = 6;                 // reset watchdog controller
v0 = CRG_TOP_PMU_CTRL_REG;
CRG_TOP_PMU_CTRL_REG = v0 & 0xFFFFFFF7;         // clear COM_SLEEP (3)
do
  ptr_sys_stat_reg = CRG_TOP_SYS_STAT_REG;
while ( !(ptr_sys_stat_reg & 0x800) == 0 );     // spin until peripheral power domain is up (3) PD_PER
GPIO_P0_01_MODE_REG = 0x200;                    // open drain input
GPIO_P0_00_MODE_REG = 0x100;                    // push-pull,input pullup,
GPIO_P0_00_MODE_REG = 0x200;                    // // open drain input
WDOG_feed_ff();
ptr_pmu_ctrl_reg = CRG_TOP_PMU_CTRL_REG;
CRG_TOP_PMU_CTRL_REG = ptr_pmu_ctrl_reg & 0xFFFFFFFB;// clear TIM_SLEEP timers sleep (2)
do
  ptr_sys_stat_reg_ = CRG_TOP_SYS_STAT_REG;
while ( !(ptr_sys_stat_reg_ & 0x200) == 0 );    // spin until the timer power domain is up
ptr_power_ctrl_reg = CRG_TOP_POWER_CTRL_REG;
CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFFFF | 0x400000;
CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F | 0x400080;
ptr_pmu_ctrl_reg = CRG_TOP_PMU_CTRL_REG;
CRG_TOP_PMU_CTRL_REG = ptr_pmu_ctrl_reg_ & 0xFFFFFFFE;
do
  ptr_sys_stat_reg__ = CRG_TOP_SYS_STAT_REG;
while ( !(ptr_sys_stat_reg__ & 8) == 0 );
OTPC_enable_clock_and_reset(1);
OTPC_set_read_mode();
QSPIC_set_manual_mode();
QSPIC_Software_Reset_peripheral();              // reset the qspi device
LOBYTE(configuration_script) = 1;
dword_2003C958 = 0;
ptr_clk_com_reg = CRG_COM_CLK_COM_REG;
CRG_COM_CLK_COM_REG = ptr_clk_com_reg | 1;      // enable UART clock
UART_reset_and_configure_uart(0x1106);          // reset and configure uart
GPIO_P0_09_MODE_REG = 2;                        // configure GPIO P0_09 function 0x2 - UART_tx
GPIO_P0_08_MODE_REG = 1;                        // configure GPIO P0_08 function 0x1 - Uart_rx
WDOG_feed_ff();
ConfigurationScript_Read((struct_configuration_script_ptr *)&configuration_script);// read the configuration scrpt from flash or qspi to address 0x2003c954
clk_ctrl_reg = CRG_TOP_CLK_CTRL_REG;
if ( (clk_ctrl_reg & 0x4000) == 0 )             // check if we are using XTAL32M clock
{
  ptr_xtal32_ctrl1_reg = CRG_XTAL_XTAL32M_CTRL1_REG;
  CRG_XTAL_XTAL32M_CTRL1_REG = ptr_xtal32_ctrl1_reg | 0x802000;// Enable xtal (startup) or enable xtal block (software 0x0 ABLE control mode) - testing only, to enable xtal, use PDC.
}
BYTE1(xtal32m_ready_flag) = 0;                  // believe this is waiting for the xtal32m to become ready and settled
TIMER_CTRL_timer_mode_set_c80(0x14);
while ( (unsigned __int8)xtal32m_ready_flag != 1 )
  ;
TIMER_CTRL_timer_mode_set_c80(0x64);
do
{
  ptr_xtal32_stat_reg = CRG_XTAL_XTAL32M_STAT1_REG;
  if ( (ptr_xtal32_stat_reg & 0x200) != 0 )
  {
    ptr_xtal32_stat1_reg = CRG_XTAL_XTAL32M_STAT1_REG;
    if ( (ptr_xtal32_stat1_reg & 0x400) != 0 )
      break;
  }
}
while ( (unsigned __int8)xtal32m_ready_flag != 1 );
if ( (unsigned __int8)xtal32m_ready_flag != 1 )
{
  BYTE1(xtal32m_ready_flag) = 1;
  TIMER_CTRL_enable_and_clear();
  TIMER_CTRL_timer_mode_set_c80(20);
  while ( (unsigned __int8)xtal32m_ready_flag != 1 )
    ;
  ptr_xtal32_ctrl0_reg = CRG_XTAL_XTAL32M_CTRL0_REG;
  CRG_XTAL_XTAL32M_CTRL0_REG = ptr_xtal32_ctrl0_reg | 0x40000000;//  enable OXTAL for the system PLL.
}
do
{
  if ( (_BYTE)configuration_script )            // if we have a configuration script
  {
    WDOG_feed_ff();
    ptr_sys_ctrl_reg = CRG_TOP_SYS_CTRL_REG;
    CRG_TOP_SYS_CTRL_REG = ptr_sys_ctrl_reg | 0x80;// enable debugger
    if ( BYTE1(xtal32m_ready_flag) )
      CLK_Set_source_xtal32m();
    get_fw_ack = UART_Get_FW((int)&configuration_script);
    CLK_Enable_RC32M();
    if ( get_fw_ack )
      RESET_to_REMAP_ADR_val(3u);               // SW_Reset to RAM - boot uart fw
  }
  QSPI_Read_Product_Headers((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
}
while ( BYTE1(configuration_script) != 1 );
WDOG_feed_ff();
if ( check_current_fw_addr_and_update_addr(product_img_offsets) )// differing offsets mean we have an update, enter
{
  if ( !SECUREBOOT_check_and_validate(
          product_img_offsets[1],
          (FW_ImageHeader *)&dword_2003C950,
          (int)product_img_offsets,
          (struct_configuration_script_ptr *)&configuration_script) )// / returns 1 if secure boot is not enabled
  {
    QSPI_process_product_update((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);// replace both of the product headers with the new one
    WDOG_Pet_1c34();
  }
}                                               // no update path
                                                // this returns 0 from [0] offset check
                                                //   if ( !(qspi_data_value_ptr[0] )
                                                //     return 0;
else if ( !SECUREBOOT_check_and_validate(
            product_img_offsets[0],
            (FW_ImageHeader *)&dword_2003C950,
            (int)product_img_offsets,
            (struct_configuration_script_ptr *)&configuration_script) )// returns 1 if secure boot is not enabled
{
  WDOG_Pet_1c34();                              // we hit this
}
SECUREBOOT_CHECK_key_revocation(dword_2003C950);
sub_A8E((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
WDOG_feed_ff();
if ( !SECUREBOOT_CHECK_setup_QSPIC_CTR_195C(
        (struct_configuration_script_ptr *)&configuration_script,
        product_img_offsets) )
  WDOG_Pet_1c34();
OTPC_standby();
Cache_setup_qspi_cache(dword_2003C950, product_img_offsets);
write_to_sysram(dword_2003C950, product_img_offsets);
return RESET_to_REMAP_ADR_val(2u);              // SW Reset to QSPI Flash
}
```

BUGS

Secureboot pain point - writable areas

Flash format defines these