1. 下载 Linaro 交叉编译工具链以及 uboot 源代码，建立编译环境，然后根据使用的芯片编译对应的版本，以下内容以 K2E 为例。
   a. 参考 http://processors.wiki.ti.com/index.php/MCSDK_UG_Chapter_Exploring 3.2.2.2 节
   b. uboot：git://git.ti.com/keystone-linux/u-boot.git 以及 http://git.ti.com/keystone-linux/u-boot
2. 修改时钟输入，根据硬件时钟输入修改各路时钟以及锁相环，配置的结构体定义在 board\ti\ks2_evm\board_k2e.c 文件中；

   首先需要修改时钟输入，需要注意的是 uboot 里目前支持的 SGMII Serdes 的时钟输入为 156.25MHz，其余的输入需要修改 Serdes 配置，建议客户均使用 156.25MHz 时钟输入；

```
unsigned int external_clk[ext_clk_count] = {
      [sys_clk]    =     100000000,
      [alt_core_clk]=    100000000,
      [pa_clk]     =     100000000,
      [ddr3_clk]   =     100000000,
      [mcm_clk]    =     312500000,
      [pcie_clk]   =     100000000,
      [sgmii_clk]  =     156250000,
      [xgmii_clk]  =     156250000,
      [usb_clk]    =     100000000
};
```

   仍然在 board_k2e.c 里，看到锁相环的配置为

```
static struct pll_init_data pll_config[] = {
      CORE_PLL_1200,
      PASS_PLL_1000,
};

#ifdef CONFIG_SPL_BOARD_INIT
static struct pll_init_data spl_pll_config[] = {
      CORE_PLL_800,
};
```

   在 Clock-k2e.h（共有四个同名文件均需修改）可以找到 PLL 的定义，将其修改为需要的值，比如输入时钟 156.25MHz 时，获得 1200MHz 主频，则需要修改 CORE_PLL_1200 为 {CORE_PLL, 384， 25， 2}， 即 156.25*384/(25*2)=1200；注意 DDR3 的配置数字表示 DDR3 的时钟频率，数据速率还应再乘以 2，如 DDR3_PLL_400 对应的是 DDR3 800MTS。

```
#define CORE_PLL_800     { CORE_PLL, 16,    1, 2 }
#define CORE_PLL_1000    { CORE_PLL, 20,    1, 2 }
#define CORE_PLL_1200    { CORE_PLL, 24,    1, 2 }
#define PASS_PLL_1000    { PASS_PLL, 20,    1, 2 }
#define DDR3_PLL_200     { DDR3_PLL, 4,     1, 2 }
#define DDR3_PLL_400     { DDR3_PLL, 16,    1, 4 }
#define DDR3_PLL_800     { DDR3_PLL, 16,    1, 2 }
#define DDR3_PLL_333     { DDR3_PLL, 20,    1, 6 }
```

打开 debug 选项（修改根目录下 config.mk 文件，DBGFLAGS=-g -DDEBUG），重新编译 uboot 并下载到 K2E 上运行，观察串口的输出，如果时钟配置正确的话，应能在串口看到类似打印，则时钟配置正确，准备进入下一步调试 DDR3。如果输出乱码，则需要检查硬件时钟是否正常，配置是否与硬件时钟输入一致等等。

```
U-Boot 2013.01 (Jun 13 2014 - 11:09:54)

U-Boot code: 0C001000 -> 0C04AD64  BSS: -> 0C0ADB08
I2C:   ready
92 13 0b 08 03 19 02 09 0b 11 01 08 0a 00 fe 00    ................
69 78 69 30 69 11 18 81 00 05 3c 3c 00 f0 83 05    ixi0i.....<<....
80 00 00 00 00 00 00 00 00 85 00 00 00 00 00 00    ................
00 00 00 00 00 00 00 00 00 00 00 00 0f 11 23 00    ..............#.
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00 00 00 00 00 80 2c 0f 13 36 e9 8e 2d 2a cc 33    ......,..6..-*.3
31 38 4b 53 46 35 31 32 37 32 48 5a 2d 31 47 36    18KSF51272HZ-1G6
4b 32 4b 32 80 2c 00 00 00 00 00 00 00 00 00 00    K2K2.,..........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff    ................
................
```

3. 修改 DDR3 初始化配置。首先确定 DDR3 的输入时钟和锁相环配置均已按照用户自己的需要正确修改；接着修改 board/ti/ks2_evm/ddr_k2e.c 相应的代码；
用户应根据自己的 DDR3 的大小配置全局描述符里的 ddr3_size 选项，如为 1GB 时，在 init_ddr3 函数内修改

```
gd->ddr3_size = 1;
```

在该文件定义了两个结构体，对应 DDR3 的 phy 和 emif 的配置寄存器，应按照使用的 DDR3 颗粒的手册对应填写，建议使用 STK 的 memory test 代码进行 DDR3 的读写测试并得到稳定的参数配置。

```
/************************ ****************************/
static struct ddr3_phy_config ddr3phy_1600_64 = {
      .pllcr            = 0x0001C000ul,
      .pgcr1_mask = (IODDRM_MASK | ZCKSEL_MASK),
```

```
        .pgcr1_val  = ((1 << 2) | (1 << 7) | (1 << 23)),
        .ptr0       = 0x42C21590ul,
        .ptr1       = 0xD05612C0ul,
        .ptr2       = 0, /* not set in gel */
        .ptr3       = 0x08861A80ul,
        .ptr4       = 0x0C827100ul,
        .dcr_mask   = (PDQ_MASK | MPRDQ_MASK | BYTEMASK_MASK),
        .dcr_val    = ((1 << 10)),
        .dtpr0          = 0x9D9CBB66ul,
        .dtpr1          = 0x12840300ul,
        .dtpr2          = 0x5002D200ul,
        .mr0        = 0x00001C70ul,
        .mr1        = 0x00000006ul,
        .mr2        = 0x00000018ul,
        .dtcr       = 0x710035C7ul,
        .pgcr2          = 0x00F07A12ul,
        .zq0cr1         = 0x0001005Dul,
        .zq1cr1         = 0x0001005Bul,
        .zq2cr1         = 0x0001005Bul,
        .pir_v1         = 0x00000033ul,
        .pir_v2         = 0x0000FF81ul,
};

static struct ddr3_emif_config ddr3_1600_64 = {
        .sdcfg          = 0x6200CE62ul,
        .sdtim1         = 0x166C9855ul,
        .sdtim2         = 0x00001D4Aul,
        .sdtim3         = 0x421DFF53ul,
        .sdtim4         = 0x543F07FFul,
        .zqcfg          = 0x70073200ul,
        .sdrfc          = 0x00001869ul,
};
```

如果未使用 ECC，则需要在 arch\arm\cpu\armv7\keystone\Ddr3.c 里修改 init_ddrphy 函数，添加关闭 ECC 的动作

```
while ((__raw_readl(base + KS2_DDRPHY_PGSR0_OFFSET) & 0x1) != 0x1)
            ;
//Modified for close ecc
__raw_writel(0x7C000E80, base + KS2_DDRPHY_DATX8_8_OFFSET);

__raw_writel(phy_cfg->pir_v2, base + KS2_DDRPHY_PIR_OFFSET);
while ((__raw_readl(base + KS2_DDRPHY_PGSR0_OFFSET) 0x1) != 0x1)
            ;
```

同时在 board_k2e.c 里注销 ECC 的初始化操作

```
//Modified for no ECC
//init_ddr3_ecc(KS2_DDR3_EMIF_CTRL_BASE);
```

最后根据 DDR3 的大小修改 include\configs\Ks2_evm.h 的定义

```
#define CONFIG_MAX_RAM_BANK_SIZE    (1 << 30)   /* 1GB */
```

4. 修改EMIF端口配置，EMIF口共有4根片选信号，根据硬件的配置修改以下结构体的参数即可

```
static struct async_emif_config
async_emif_config[ASYNC_EMIF_NUM_CS] = {
	{	/* CS0 */
		.mode		= ASYNC_EMIF_MODE_NAND,
		.wr_setup	= 0xf,
		.wr_strobe	= 0x3f,
		.wr_hold	= 7,
		.rd_setup	= 0xf,
		.rd_strobe	= 0x3f,
		.rd_hold	= 7,
		.turn_around= 3,
		.width		= ASYNC_EMIF_8,
	},
};
```

至此board_init_f里的初始化配置结束，uboot代码进行relocation之后进入DDR3里运行，开始board_init_r的操作；

5. 在 EVM 的配置中，board_init_r 需要修改以太网的配置，首先在 board_k2e.c 里修改每一个 SGMII 端口的配置，需要注意的是 Serdes 配置需要按照 TI 在 MCSDK 里提供的配置进行，uboot 的代码里默认使用了 156.25MHz 的输入时钟，SGMII 配置成 1000M 的模式，建议客户按此模式选择 PHY 芯片；对于 SGMII 端口与 PHY 芯片的通信，一般采用自协商的模式，首先确保 PHY 芯片正常工作，可以在 MDIO 端口读取 PHY 的寄存器看是否正常，K2 的 MDIO 端口电压为 1.8V，其它的电压需要做电平转换。uboot 里会检测 serdes 链路和 MDIO 链路的工作状态，如果 SGMII 端口不能正常连接，建议打印如下寄存器的值 0x02325FF4, 0x02325FE0 ~ 0x02325FEC, 0x0232BFF4, 0x0232BFE0 ~ 0x0232BFEC；可以通过这些寄存器检查某一路 Serdes 的信号，如果发现错误，通常需要检查 K2 的 SGMII 时钟输入和 PHY 芯片的工作状态。

```
eth_priv_t eth_priv_cfg[] =
{
	{
		.int_name	= "K2E_EMAC0",
		.rx_flow	= CPSW_PORT_RX_FLOW(0),
		.phy_addr	= 0,
		.slave_port = 1,
		.sgmii_link_type = SGMII_LINK_MAC_PHY,
	},
	…
}
```

board_init_r结束后，系统进入命令行等待用户输入，uboot至此完成初始化的工作。

重新编译uboot并下载，输出的log为

U-Boot 2013.01 (Jun 13 2014 - 11:09:54)

U-Boot code: 0C001000 -> 0C04AD64 BSS: -> 0C0ADB08
I2C: ready
.....
.....
Detected SO-DIMM [18KSF51272HZ-1G6K2]
DRAM: 1 GiB
ram_size is 1073741824
Reseting entire DDR3 memory to 0 ...
monitor len: 000ACB08
ramsize: 40000000
TLB table from bfff0000 to bfff4000
Top of RAM usable for U-Boot at: bfff0000
Reserving 690k for U-Boot at: bff43000
Reserving 2304k for malloc() at: bfd03000
Reserving 32 Bytes for Board Info at: bfd02fe0
Reserving 128 Bytes for Global Data at: bfd02f60
New Stack Pointer is: bfd02f50
RAM Configuration:
Bank #0: 80000000 1 GiB
relocation Offset is: b3f42000
monitor flash len: 0005252C
Now running in RAM - U-Boot at: bff43000
NAND: 512 MiB
Net: K2E_EMAC0, K2E_EMAC1, K2E_EMAC2, K2E_EMAC3, K2E_EMAC4, K2E_EMAC5, K2E_EMAC6, K2E_EMAC7
Hit any key to stop autoboot: 0
K2E EVM #