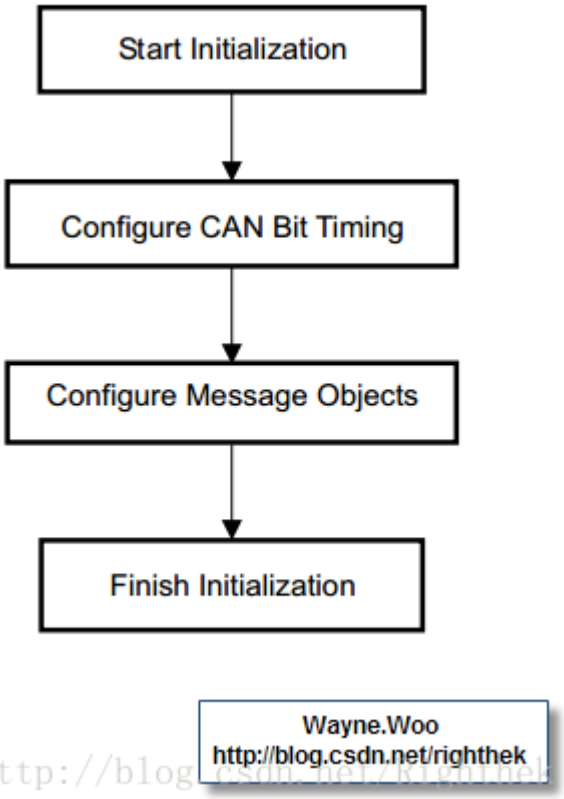
对于一般的 CAN 模块，进行初始化时，最关键的是以下两步：

1、 配置 CAN 的位时序；

2、 配置 CAN 的消息报文；




下面，我们来详细分析上面提到的关键两步。

一、初始化步骤：

1、 第一步，进入初始化模式，在 CAN 控制寄存器中，将 Init 位置 1；

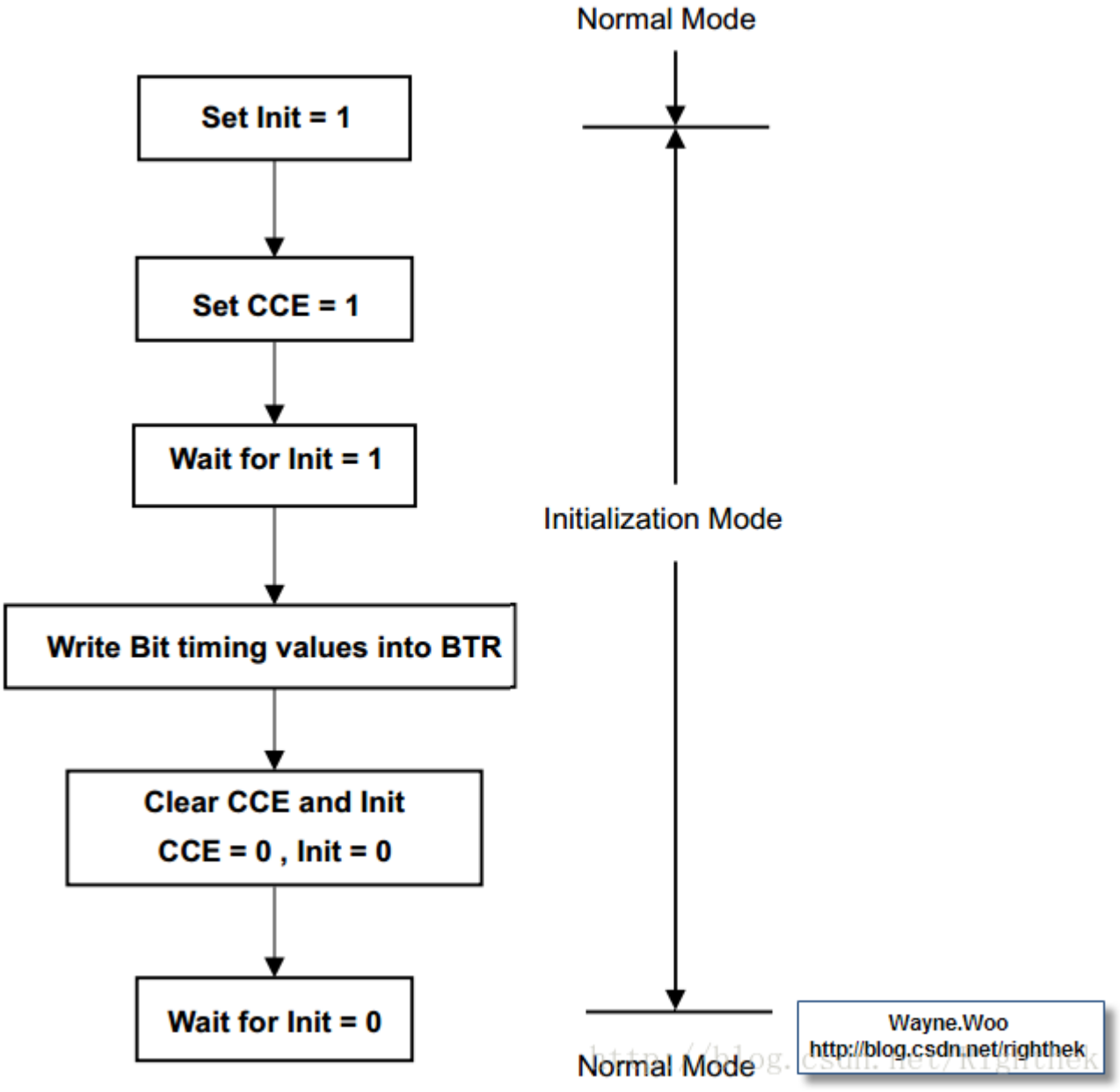
2、 第二步，在 CAN 控制寄存器中，将 CCE 位置 1；

3、 第三步，等待 Init 位置 1，此步聚为了确保已经进入初始化模式；

4、 第四步，将位时序的值写入到位时序寄存器（BTR）中；

5、 第五步，将 CCE 和 Init 位置为 0；

6、 第六步，等待清除 Init 位，此步聚为了确保已经退出初始化模式；




解释完 CAN 的初始化步骤后，我们来看看代码实现：

我们先初始化一个 CAN 的位时序常量。

```
/* CAN Bittiming constants as per D_CAN specs */
```

```
static structcan_bittiming_const d_can_bittiming_const = {
        .name = D_CAN_DRV_NAME,
        .tseg1_min = 1,              /* Time segment 1 = prop_seg + phas
e_seg1 */
        .tseg1_max = 16,
        .tseg2_min = 1,              /* Time segment 2 = phase_seg2 */
        .tseg2_max = 8,
        .sjw_max = 4,
        .brp_min = 1,
        .brp_max = 1024,    /* 6-bit BRP field + 4-bit BRPE field*/
        .brp_inc = 1,
};
```

在打开 CAN 设置时，进行初始化。初始化完成之后，恢复正常模式，使能收发 I/O 控制引脚，最后配置消息报文。代码如下：

```
static void d_can_init(structnet_device *dev)
{
        struct d_can_priv *priv =netdev_priv(dev);
        u32 cnt;

        netdev_dbg(dev, "resetting d_can...\n");
```

```
        d_can_set_bit(priv, D_CAN_CTL,D_CAN_CTL_SWR);

        /* Enterinitialization mode by setting the Init bit */
        d_can_set_bit(priv, D_CAN_CTL,D_CAN_CTL_INIT);

        /* enableautomatic retransmission */
        d_can_set_bit(priv, D_CAN_CTL,D_CAN_CTL_ENABLE_AR);

        /* Set theConfigure Change Enable ( CCE) bit */
        d_can_set_bit(priv, D_CAN_CTL,D_CAN_CTL_CCE);

        /* Wait forthe Init bit to get set */
        cnt = D_CAN_WAIT_COUNT;
        while (!d_can_get_bit(priv, D_CAN_CTL,D_CAN_CTL_INIT) && cnt !
= 0) {
                --cnt;
                udelay(10);
        }

        /* setbittiming params */
```

```
        d_can_set_bittiming(dev);

         d_can_clear_bit(priv, D_CAN_CTL,D_CAN_CTL_INIT | D_CAN_CTL_
CCE);

        /* Wait for the Init bit to get clear*/
        cnt = D_CAN_WAIT_COUNT;
        while (d_can_get_bit(priv, D_CAN_CTL,D_CAN_CTL_INIT) && cnt !
= 0) {
                --cnt;
                udelay(10);
        }

        if (priv->can.ctrlmode &(CAN_CTRLMODE_LOOPBACK |
                                CAN_CTRLMODE_LISTENONLY))
                d_can_test_mode(dev);
        else
                /* normal mode*/
                d_can_write(priv, D_CAN_CTL,D_CAN_CTL_EIE | D_CAN_CTL
_IE1 |
                                                D_CAN_CTL_IE0);
```

```
        /* Enable TXand RX I/O Control pins */
        d_can_write(priv, D_CAN_TIOC,D_CAN_TIOC_FUNC);
        d_can_write(priv, D_CAN_RIOC, D_CAN_RIOC_FUNC);

        /* configuremessage objects */
        d_can_configure_msg_objects(dev);

        /* set a LECvalue so that we can check for updates later */
        d_can_write(priv, D_CAN_ES,LEC_UNUSED);
    }
```

执行完第一、第二步之后，设置位时序的参数，代码如下：

```
    static intd_can_set_bittiming(struct net_device *dev)
    {
        struct d_can_priv *priv =netdev_priv(dev);
        const struct can_bittiming *bt =&priv->can.bittiming;
        u32 can_btc;

        can_btc = ((bt->phase_seg2 - 1)& 0x7) << D_CAN_BTR_TSEG2_S
HIFT;
```

```
            can_btc |= ((bt->phase_seg1 +bt->prop_seg - 1)
                        & 0xF) <<D_CAN_BTR_TSEG1_SHIFT;

            can_btc |= ((bt->sjw - 1) & 0x3)<< D_CAN_BTR_SJW_SHIFT;

            /* Ten bitscontains the BRP, 6 bits for BRP and upper 4 bits for b
rpe*/
            can_btc |= ((bt->brp - 1) &0x3F) << D_CAN_BTR_BRP_SHIFT;
            can_btc |= ((((bt->brp - 1) >>6) & 0xF) << D_CAN_BTR_BRPE_SH
IFT);

            d_can_write(priv, D_CAN_BTR, can_btc);

            netdev_info(dev, "setting CAN BT =%#x\n", can_btc);

            return 0;
        }
```

以上内容并不是 Linux CAN 的驱动初始化过程，只是属于其中的一部分。更具体地说，是针对 CAN 控制器的初始化过程。