# Linux DM9051 Driver r2503_v3.9.x user's guide for ptp

# Add on to original dm9051a driver

2. dm9051_plug.h

```
#define PLUG_PTP_1588
#ifdef PLUG_PTP_1588
#define DMPLUG_PTP //(ptp 1588)
#endif

** Optional configuration, by define DMPLUG_PTP.
** So that to add dm9051_ptpd.o in Makefile
** If not define DMPLUG_PTP, dm9051_ptpd.o can be omitted.
```

1. dm9051.h

```
struct board_info
{
    /* ptpc */
    #if 1 //0
    int         ptp_enable;
    int         ptp_on; //_1588_
    struct ptp_clock        *ptp_clock;
    struct ptp_clock_info   ptp_caps;

    u8          ptp_mode; //1: one-step, 2: two-step, 3: xxx, others: Not PTP
    s64         pre_rate;
    struct hwtstamp_config  tstamp_config;
    u8          rxTSbyte[8]; //_1588_
    #endif

** Operation using varibles.
```

2. dm9051.c

```
static int dm9051_probe(struct spi_device *spi)
{
    db->ptp_enable = 1;
    db->ptp_on = 0;
    dm9051_ptp_init(db); //_1588_

** Init ptp module.
```

```
** Note: db->ptp_enable = 1;
** Note: db->ptp_on = 0;
```

```
3. dm9051.c
static int dm9051_drv_remove(struct spi_device *spi)
{
    dm9051_ptp_stop(db); //_1588_ todo

** Remove stop ptp module.
```

```
4. dm9051.c
static const struct ethtool_ops dm9051_ethtool_ops = {
    .get_ts_info = dm9051_ts_info, //_1588_,

** Time stamp support info.
** Use $ ethtool -T eth1, to show eth1 ptp clock info, include clock index and
    support ts modes.
```

```
5. dm9051.c
static const struct net_device_ops dm9051_netdev_ops = {
{
    .ndo_eth_ioctl = dm9051_ptp_netdev_ioctl, //_1588_

** Time stamp config info.
```

```
6. dm9051.c
static int dm9051_loop_tx(struct board_info *db)
{
    while (!skb_queue_empty(&db->txq))
    {
        struct sk_buff *skb = skb_dequeue(&db->txq);
        if (skb) {
            db->ptp_mode = (int) dm9051_ptp_one_step(skb); //_1588_,
            db->tcr_wr = dm9051_tcr_wr(skb, db); //_1588_,
            ret = TX_PACKET(db, skb, data_len);
            dm9051_hwtstamp_to_skb(skb, db); //_1588_,
            ndev->stats.tx_packets++;
        }

** Tx tstamp processing, send tstamp.
** Tx tstamp processing, report tstamp.
```

7. dm9051.c

```
// 06H RX Status Reg
// BIT(5),PTP use the same bit, timestamp is available
// BIT(3),PTP use the same bit, this is odd parity rx TimeStamp
// BIT(2),PTP use the same bit: 1 => 8-bytes, 0 => 4-bytes, for timestamp length
#define RSR_RXTS_EN     BIT(5)
#define RSR_RXTS_PARITY     BIT(3)
#define RSR_RXTS_LEN        BIT(2)
#define RSR_PTP_BITS        (RSR_RXTS_EN | RSR_RXTS_PARITY | RSR_RXTS_LEN)

static int rx_head_break(struct board_info *db)
{
    u8 err_bits = RSR_ERR_BITS;
    err_bits &= ~RSR_PTP_BITS;

** Rx head status error bits check.
** Rx head status BIT(5),BIT(3),BIT(2) are not error bits for ptp module is enable.

static int dm9051_loop_rx(struct board_info *db)
{
    /* receive rx_tstamp */
    ret = dm9051_read_ptp_tstamp_mem(db, db->rxTSbyte);
    if (ret)
        return ret;

    padlen = (dm9051_modedata->skb_wb_mode && (rxlen & 1)) ? rxlen + 1 : rxlen;
    skb = dev_alloc_skb(padlen);

    rdptr = skb_put(skb, rxlen - 4);
    ret = dm9051_read_mem_cache(db, DM_SPI_MRCMD, rdptr, padlen);

    skb->protocol = eth_type_trans(skb, db->ndev);

    //So when NOT T1/T4, we can skip tell an empty (virtual) tstamp
    //if (db->rxhdr.status & RSR_RXTS_EN) { // Is it inserted Timestamp?
        dm9051_ptp_rx_hwtstamp(db, skb, db->rxTSbyte); //_1588_,
        /* following, wuth netif_rx(skb),
         * slave4l can parse the T1 and/or T4 rx tstamp from master
         */
    //}
    netif_rx(skb);

    db->ndev->stats.rx_packets++;
```

```
** Rx, Depend on RSR_RXTS_EN(i.e. BIT(5)) to read rx ptp tstamp mem.
** Rx, Depend on RSR_RXTS_EN(i.e. BIT(5)) to report rx ptp tstamp.
```

8. dm9051.c

9. dm9051.c

10. dm9051.c

11. dm9051_ptpd.c

```c
int dm9051_hwtstamp_to_skb(struct sk_buff *skb, struct board_info *db)
{
    /* Process PTP message based on type */
    /* TxSync one step chip-insert-tstamp, and NOT report HW tstamp to master4l
     * (master do)
     * TxSync two step chip-NOT-tstamp, but report HW timestamp to master4l
     * (master do/ master4l go ahead to do followup)
     * TxDelayReq one-step/two-step chip-NOT-insert-tstamp
     * (slave do)
     * TxDelayReq one-step/two-step report HW timestamp to slave4l
     * (slave do)
     * TxDelayResp CAN had T4 on recv DelayReq to be added with DelayResp feedback
     * to slave4l
     * (master do)
     */
    switch (message_type) {
        case PTP_MSGTYPE_SYNC:
            /* Only report HW timestamp for two-step sync */
            if (db->ptp_mode == PTP_TWO_STEP) {
                dm9051_ptp_tx_hwtstamp(db, skb);
            }
            break;
        case PTP_MSGTYPE_DELAY_REQ:
            dm9051_ptp_tx_hwtstamp(db, skb);
            break;
        default:
            netdev_dbg(ndev, "Unhandled PTP message type: 0x%02x\n", message_type);
            break;
```

```
    }

static void dm9051_ptp_tx_hwtstamp(struct board_info *db, struct sk_buff *skb)
{
    struct skb_shared_hwtstamps shhwtstamps;
    u64 ns;
    sec = s_lo;
    sec |= s_hi << 16;

    ns = ns_lo;
    ns |= ns_hi  << 16;
    ns += ((u64)sec) * 1000000000ULL;
    shhwtstamps.hwtstamp = ns_to_ktime(ns);
    //skb_complete_tx_timestamp(skb, &shhwtstamps);
    skb_tstamp_tx(skb, &shhwtstamps); //For report T3 HW tx tstamp

void dm9051_ptp_rx_hwtstamp(struct board_info *db, struct sk_buff *skb, u8
*rxTSbyte)
{
    struct skb_shared_hwtstamps *shhwtstamps;

    sec = s_lo;
    sec |= s_hi << 16;

    ns = ns_lo;
    ns |= ns_hi  << 16;

    ns += ((u64)sec) * 1000000000ULL;
    shhwtstamps = skb_hwtstamps(skb); //for pass T2 the HW rx tstamp
    memset(shhwtstamps, 0, sizeof(*shhwtstamps));
    shhwtstamps->hwtstamp = ns_to_ktime(ns);

** Tx, dm9051_hwtstamp_to_skb to report tx ptp tstamp.
** Rx, dm9051_ptp_rx_hwtstamp parse read rx ptp tstamp mem and report tstamp.
```

```
12. dm9051_ptp.c
int dm9051_ts_info(struct net_device *net_dev, struct kernel_ethtool_ts_info *info)
{
int dm9051_ptp_netdev_ioctl(struct net_device *ndev, struct ifreq *rq, int cmd) {

** Ts, callback function support.
** Config, callback function support.
```

13. dm9051_ptp.c

```
static struct ptp_clock_info ptp_dm9051a_info = {
    .owner = THIS_MODULE,
    .name = "DM9051A PTP",
    .max_adj = 50000000,
    .n_alarm = 0,
    .n_ext_ts = 0,
    .n_per_out = 0, //n_periodic_outputs
    //.n_pins = 0, //1; //n_programable_pins
    .pps = 0, //1, //0,
    .adjfine = ptp_9051_adjfine,
    .adjtime = ptp_9051_adjtime,
    .gettime64 = ptp_9051_gettime,
    .settime64 = ptp_9051_settime,
    .enable = ptp_9051_feature_enable,
    .verify = ptp_9051_verify_pin,

};

int ptp_9051_gettime(struct ptp_clock_info *ptp,
    struct timespec64 *ts)
{

int ptp_9051_settime(struct ptp_clock_info *ptp,
    const struct timespec64 *ts)
{

** define .name = "DM9051A PTP",
** define .max_adj = 50000000,
** define .pps = 0,
** Subroutines support for struct ptp_clock_info of ptp's callback hook functions.
** ptp_9051_gettime, mutex_lock(&db->spi_lockm) and mutex_unlock(&db->spi_lockm)
** ptp_9051_settime, mutex_lock(&db->spi_lockm) and mutex_unlock(&db->spi_lockm)
```

14. dm9051_ptp.c

```
int ptp_9051_adjfine(struct ptp_clock_info *ptp, long scaled_ppm)
{

int ptp_9051_adjtime(struct ptp_clock_info *ptp, s64 delta)
{

** ptp_9051_adjfine, mutex_lock(&db->spi_lockm) and mutex_unlock(&db->spi_lockm)
** ptp_9051_adjtime, mutex_lock(&db->spi_lockm) and mutex_unlock(&db->spi_lockm)
```

```
15. dm9051_ptp.c
16. u32 dm9051_get_rate_reg(struct board_info *db);
int dm9051_read_ptp_tstamp_mem(struct board_info *db, u8 *rxTSbyte); //for rx

** Subroutines support functions.
```

```
16. dm9051_ptp.c
static unsigned int ptp_packet_classify(struct sk_buff *skb);
static int is_ptp_sync_packet(struct ptp_header *hdr, unsigned int ptp_class);

** Static subroutines support functions.
```

```
17. dm9051_ptp.c
enum ptp_sync_type dm9051_ptp_one_step(struct sk_buff *skb, struct board_info *db);
unsigned int dm9051_tcr_wr(struct sk_buff *skb, struct board_info *db); //for tx

** TX support subroutines functions.
** In dm9051_ptp_one_step,
   it return below:
    db->ptp_mode = 0/1/2/3;
   it parses to has below,
    db->ptp_packet = 0/1;
    db->ptp_sync = 0/1;
    db->ptp_step = PTP_TWO_STEP/PTP_ONE_STEP;
   Note: Driver use according db->ptp_mode to execute following operations.
     It could be changed to use according db->ptp_packet, db->ptp_sync,
     db->ptp_step instead.
```

```
18. dm9051_ptp.c
void dm9051_ptp_init(struct board_info *db);
void dm9051_ptp_stop(struct board_info *db);

** dm9051_ptp_init, do ptp_clock_register, and dm9051a ptp initialize
** dm9051_ptp_stop, do ptp_clock_unregister, and dm9051a ptp clock disable
```

## Key Note

1.Explicity list every items related to dm9051a ptp, base on dm9051a original standard driver.
2.db->ptp_mode = (int) dm9051_ptp_one_step(skb), The usage of db->ptp_mode could be changed by use db->ptp_packet, db->ptp_sync, and db->ptp_step instead.
3.Example execution result of $ ethtool -T eth1

```
Time stamping parameters for eth1:
    Capabilities:
            hardware-transmit
            software-transmit
            hardware-receive
            software-receive
            software-system-clock
            hardware-raw-clock
    PTP Hardware Clock: 1
    Hardware Transmit Timestamp Modes:
            off
            on
            onestep-sync
    Hardware Receive Filter Modes:
            none
            all
```