

## Overview

This application note provides example code to dump the content of the DS31256 registers, queues, descriptors and FIFO RAM's to a specified file. This data is helpful when the DS31256 is not working correctly. It provides important information for further investigation and debugging.

For example, the setting of every DS31256 register will be shown in the file after the register dump processes. The configuration is the first place that needs examination to make sure the device has been configured correctly. It will be requested by the support team when any problem occurs. Sometimes, the problem can be resolved simply by checking the configuration registers.

This application note only lists the most common area that may cause problems. The content of the register dump file provides information about the configuration of the registers, queues, descriptors and FIFO RAM's.

Before performing a register dump, there are some assumptions. The following functions are needed in the software.

1. `write_reg (addr, data)` -- writes the specified data value to the specified DS31256 register  
    addr = DS31256 register offset from the chip base address  
    data = data to be written to register
2. `read_reg (addr)` -- reads the DS31256 register at the specified address and returns the value  
    addr = DS31256 register offset from the chip base address
3. `read_ind_reg (addr, i)` -- read the DS31256 indirect register at the specified address and returns the value  
    addr = DS31256 register offset from the chip base address  
    i = index
4. The standard C functions `printf`, `fopen`, `fprintf`, and `fclose`
5. Some data structures will be used to hold the data in the queues and descriptors

### **1. Example code to dump the DS31256 registers data to a specified file**

```

void dumpRegs (char *filename)
{
    int addr, i, port;
    FILE *fptr;

    /* Open specified file for register dump */
    fptr = fopen (filename, "w");

    if (fptr == NULL)
    {
        printf ("Can't open the file\n");
        return;
    }

    /* Dump the contents of registers by category to the specified file */
    fprintf (fptr, " Dump of DS31256 Registers\n");

    fprintf (fptr, " General Config Registers\n");
    fprintf (fptr, "MRID:\t % .4x\n", read_reg(0x0));
    fprintf (fptr, "MC: \t % .4x\n", read_reg(0x10));
    fprintf (fptr, "SM: \t % .4x\n", read_reg(0x20));
    fprintf (fptr, "ISM: \t % .4x\n", read_reg(0x24));
    fprintf (fptr, "SDMA: \t % .4x\n ", read_reg(0x28));
    fprintf (fptr, "ISDMA: \t % .4x\n ", read_reg(0x2C));
    fprintf (fptr, "\tSV54:\t % .4x\n", read_reg(0, 0x30));
    fprintf (fptr, "ISV54: \t % .4x\n", read_reg(0x34) );
    fprintf (fptr, "LBBMC: \t % .4x\n ", read_reg(0x40));
    fprintf (fptr, "TEST: \t % .4x\n", read_reg(0x50));

    fprintf (fptr, "Receive Port Registers \n");
    for (i = 0, addr = 0x100; addr <= 0x13c; addr += 4, i++)
    {
        fprintf (fptr, "RP %d CR: \t % .4x\n", i, read_reg(addr));
        if ((i & 3) == 3)
            fprintf (fptr, "\n");
    }

    fprintf (fptr, "Transmit Port Registers \n");
    for (i = 0, addr = 0x200; addr <= 0x23c; addr += 4, i++)
    {
        fprintf (fptr, "TP%d CR:\t % .4x", i , read_reg( addr));
        if ((i & 3) == 3)
            fprintf (fptr, "\n");
    }
}

```

```

for (port = 0, addr = 0x300; port < 16; port++, addr += 8)
{
    fprintf (fptr, "Channelized Port Registers, Port %d \n", port);
    for (i = 0; i < 128; i++)
    {
        fprintf (fptr,"C%d DAT%d : \t %.4x", port, i , read_ind_reg(addr, i));
        fprintf (fptr, "R%dCFG%d : \t %.4x", port, i ,read_ind_reg(addr, i | 0x100));
        fprintf (fptr, "T%dCFG%d: \t %.4x", port, i, read_ind_reg(addr, i | 0x200));
        printf (fptr, "\n");
    }

    fprintf (fptr, "HDLC Registers \n");
    fprintf (fptr, "RHPL: \t %.4x\n", read_reg(0x410));

    for (i = 0; i < 256; i++)
    {
        fprintf (fptr,"RHCD%d : \t %.4x", i, read_ind_reg (0x400, i));
        fprintf (fptr,"THCD%d : \t %.4x", i, read_ind_reg (0x480, i));
        fprintf (fptr, "\n");
    }
}

fprintf (fptr, "BERT Registers\n");
fprintf (fptr, "BERTC0 : \t %.4x", read_reg(0x500));
fprintf (fptr, "BERTC1 : \t %.4x", read_reg(0x504));
fprintf (fptr, "BERTRP0 : \t %.4x", read_reg(0x508));
fprintf (fptr, "BERTRP1 : \t %.4x \n", read_reg(0x50c));
fprintf (fptr, "BERTBC0 : \t %.4x \n", read_reg(0x510));
fprintf (fptr, "BERTBC1 : \t %.4x \n", read_reg(0x514));
fprintf (fptr, "BERTEC0 : \t %.4x \n", read_reg(0x518));
fprintf (fptr, "BERTEC1 : \t %.4x \n", read_reg(0x51c));

fprintf (fptr,"Receive DMA Registers \n");
fprintf (fptr,"RFQBA0 : \t %.4x", read_reg(0x700));
fprintf (fptr,"RFQBA1 : \t %.4x", read_reg(0x704));
fprintf (fptr,"RFQEA : \t %.4x", read_reg(0x708));
fprintf (fptr,"RFQSBSA : \t %.4x \n", read_reg(0x70c));
fprintf (fptr,"RFQLBWP : \t %.4x", read_reg(0x710));
fprintf (fptr,"RFQSBWP : \t %.4x", read_reg(0x714));
fprintf (fptr,"RFQLBRP : \t %.4x", read_reg(0x718));
fprintf (fptr,"RFQSBRP : \t %.4x", read_reg(0x71c));
fprintf (fptr,"RDQBA0 : \t %.4x", read_reg(0x730));
fprintf (fptr,"RDQBA1 : \t %.4x", read_reg(0x734));
fprintf (fptr,"RDQEA : \t %.4x", read_reg(0x738));
fprintf (fptr,"RDQRP : \t %.4x \n", read_reg(0x73c));

```

```

fprintf (fptr,"RDQWP : \t %.4x", read_reg(0x740));
fprintf (fptr,"RDQFFT : \t %.4x", read_reg(0x744));
fprintf (fptr,"RDBA0 : \t %.4x", read_reg(0x750));
fprintf (fptr,"RDBA1 : \t %.4x \n", read_reg(0x754));
fprintf (fptr,"RDMAQ : \t %.4x", read_reg(0x780));
fprintf (fptr,"RLBS : \t" %.4x", read_reg(0x790));
fprintf (fptr,"RSBS : \t" %.4x \n", read_reg(0x794));

fprintf (fptr,"Receive DMA Channel Configuration \n");
for (i = 0; i < 256; i++)
{
    fprintf (fptr,"Channel %d : \t ", i);
    fprintf (fptr, "%.4x", read_ind_reg (0x770, 0x100 + i));
    fprintf (fptr, "%.4x \t", read_ind_reg (0x770, 0x000 + i));
    fprintf (fptr, "%.4x", read_ind_reg (0x770, 0x300 + i));
    fprintf (fptr, "%.4x \t", read_ind_reg (0x770, 0x200 + i));
    fprintf (fptr, "%.4x", read_ind_reg (0x770, 0x500 + i));
    fprintf (fptr, "%.4x \n", read_ind_reg ( 0x770, 0x400 + i));
}

```

```

fprintf (fptr, "Transmit DMA Registers \n");
fprintf (fptr, "TPQBA0 : \t %.4x", read_reg(0x800));
fprintf (fptr, "TPQBA1 : \t %.4x", read_reg(0x804));
fprintf (fptr, "TPQEA : \t %.4x", read_reg(0x808));
fprintf (fptr, "TPQWP : \t %.4x", read_reg(0x80c));
fprintf (fptr, "TPQRP : \t %.4x \n", read_reg(0x810));
fprintf (fptr, "TDQBA0 : \t %.4x", read_reg(0x830));
fprintf (fptr, "TDQBA1 : \t %.4x", read_reg(0x834));
fprintf (fptr, "TDQEA : \t %.4x", read_reg(0x838));
fprintf (fptr, "TDQWP : \t %.4x", read_reg(0x83c));
fprintf (fptr, "TDQRP : \t %.4x \n", read_reg(0x840));
fprintf (fptr,"TDQFFT : \t %.4x", read_reg(0x844));
fprintf (fptr,"TDBA0 : \t %.4x", read_reg(0x850));
fprintf (fptr, "TDBA1 : \t %.4x", read_reg(0x854));
fprintf (fptr, "TDMAQ : \t %.4x \n", read_reg(0x880));

```

```

fprintf (fptr, "Transmit DMA Channel Configuration \n");
for (i = 0; i < 256; i++)
{
    fprintf (fptr, "Channel %d : \t", i);
    fprintf (fptr, "%.4x", read_ind_reg(0x870, 0x100 + i));
    fprintf (fptr, "%.4x \t", read_ind_reg(0x870, 0x000 + i));
    fprintf (fptr, "%.4x", read_ind_reg(0x870, 0x300 + i));
    fprintf (fptr, "%.4x \t", read_ind_reg(0x870, 0x200 + i));
    fprintf (fptr, "%.4x", read_ind_reg(0x870, 0x500 + i));
}

```

```

fprintf (fptr, "%.4x \n", read_ind_reg(0x870, 0x400 + i));
fprintf (fptr, "%.4x", read_ind_reg(0x870, 0x700 + i));
fprintf (fptr, "%.4x \t", read_ind_reg(0x870, 0x600 + i));
fprintf (fptr, "%.4x", read_ind_reg(0x870, 0x900 + i));
fprintf (fptr, "%.4x \t", read_ind_reg(0x870, 0x800 + i));
fprintf (fptr, "%.4x", read_ind_reg(0x870, 0xb00 + i));
fprintf (fptr, "%.4x \n", read_ind_reg(0x870, 0xa00 + i));
}

```

## 2. Example code to dump the data of transmit queues and descriptors to a specified file.

Some structures are used to store the data of pending queues, done queues and descriptors. This data can help to check the data has been created and transmit correctly.

```

void dumpTx(drvDev * dp, int32 filename, int32 channel, int32 desc2)
{
    FILE fptr;
    dmaTxDoneQDesc doneq;    /* structure of Transmit Done Queue */
    dmaTxDev tdma;          /* structure of DMA TX subsystem of the device */
    dmaTxBufDesc tbd;       /* structure of Transmit Buffer Descriptor */
    dmaPendQDesc * pendq;   /* structure of Transmit Pending Queue*/

    int i, a = 0;
    int32 dev = 0;
    int32 tdqrp, tdqwp;
    int32 tpqrp, tpqwp;
    tdma = &dp->txDma;      /* Get the pointer from the device structure */
    doneq = tdma->doneQStart; /* Transmit Done Queue Start Pointer */
    pendq = tdma->pendQStart; /* Transmit Pending Queue Start Pointer */

    /* Open the specified file for data dump */
    fptr = fopen(filename, "w");

    if (fptr == NULL)
    {
        printf ("Can't open the file\n");
        return;
    }

    fprintf (fptr, "\nTx Buffer Descriptors : \n");
    for(i=0; i<tdma->bufDescCnt; i++)
    {
        tbd = &tdma->bufDesc[i];
        fprintf (fptr, "%4x : %.8x %.8x %.8x %.8x\n", i, tbd->dataAddr, tbd->desc1,
                tbd->desc2,

```

```

        tbd->desc3);
    }

    /* Check the transmit pending queue write and read pointer before print out the data */
    tpqwp = read_reg (0x80c);
    tpqrp = read_reg (0x810);

    fprintf (fptr, "\nTransmit Pending Queues : \n");
    fprintf (fptr, "Read Pointer: %.8x Write Pointer : %.8x\n",tpqrp,tpqwp);
    pendq = tdma->pendQStart;

    for(i = pendq; i <= tdma->pendQEnd; pendq++)
    {
        fprintf (fptr, "%4x : %.8x\n", a, pendq->desc);
        a++;

        if (pendq == tdma->pendQEnd)
        {
            break;
        }
    }

    /* Check the transmit done queue write and read pointer before print out the data */
    tdqwp = read_reg(0x840);
    tdqrp = read_reg(0x83c);

    fprintf (fptr, "\nTransmit Done Queue Descriptors : \n");
    fprintf (fptr, "Read Pointer: %.8x Write Pointer : %.8x\n",tdqrp,tdqwp);
    a=0;
    doneq = tdma->doneQStart;
    for (i = doneq; i <= tdma->doneQEnd; doneq++)
    {
        fprintf (fptr,"%4x : %.8x\n", a, doneq->desc);
        a++;
        if (doneq == tdma->doneQEnd)
        {
            break;
        }
    }

    /* Close the file */
    fclose (fptr);
}

```

### 3. Example code to dump the Receive FIFO RAM's to a specified file

```
void dump_RX_FIFO_RAMs(char *filename)
{
    unsigned long reg_val;
    unsigned long rftst1, rftst2, rftst3, rftst4, rftst5, rftst6;
    unsigned long ram_addr;
    unsigned long max_busy_cnt, busy_cnt;
    FILE *fptr;

    /* DS31256 RX FIFO test register addresses */
    rftst1 = 0x09D0;
    rftst2 = 0x09D4;
    rftst3 = 0x09D8;
    rftst4 = 0x09DC;
    rftst5 = 0x09E0;
    rftst6 = 0x09E4;

    max_busy_cnt = 10;
    printf ("Dumping RX FIFO RAM's to file %s\n\n", filename);
    /* Open specified file for RAM data dumping */
    fptr = fopen(filename, "w");

    /* Put chip in test mode */
    write_reg(0x0050, 0x0001);

    /* Dump RX FIFO SBP RAM */
    /* Will be covered by write configuration and read configuration RAM dumps below */

    /* Dump RX FIFO BP RAM to file */
    fprintf (fptr, "RX FIFO BP RAM\n");
    for (ram_addr=0; ram_addr<1024; ram_addr++)
    {
        fprintf (fptr, "%.8x : ", ram_addr);
        write_reg(rftst1, (0x0400 + ram_addr) | 0x4000);
        busy_cnt = 0;
        while((read_reg(rftst1) & 0x8000) && (busy_cnt++ < max_busy_cnt));
        if (busy_cnt >= max_busy_cnt)
        {
            printf ("\n\nERROR: Max busy cnt of %d on rftst1 - RX FIFO BP RAM\n",
                max_busy_cnt);
            fclose (fptr);
            return;
        }
        fprintf (fptr, "%.3x\n", read_reg(rftst2) & 0x03ff);
    }
}
```

```

}
fprintf (fptr, "\n");

/* Dump RX FIFO HWM RAM to file */
fprintf (fptr, "RX FIFO HWM RAM\n");
for (ram_addr=0; ram_addr<256; ram_addr++)
{
    fprintf (fptr, "%.8x : ", ram_addr);
    write_reg(rftst1, (0x0800 + ram_addr) | 0x4000);
    busy_cnt = 0;
    while((read_reg(rftst1) & 0x8000) && (busy_cnt++ < max_busy_cnt));
    if (busy_cnt >= max_busy_cnt)
    {
        printf ("\n\nERROR: Max busy cnt of %d on rftst1 - RX FIFO HWM RAM\n",
            max_busy_cnt);
        fclose (fptr);
        return;
    }
    fprintf (fptr, "%.3x\n", read_reg(rftst2) & 0x03ff);
}
fprintf (fptr, "\n");

/* Dump RX FIFO write configuration RAM to file */
fprintf (fptr, "RX FIFO Write Configuration RAM\n");
for (ram_addr=0; ram_addr<256; ram_addr++)
{
    fprintf (fptr, "%.8x : ", ram_addr);
    write_reg(rftst1, (0x0c00 + ram_addr) | 0x4000);
    busy_cnt = 0;
    while((read_reg(rftst1) & 0x8000) && (busy_cnt++ < max_busy_cnt));
    if (busy_cnt >= max_busy_cnt)
    {
        printf ("\n\nERROR: Max busy cnt of %d on rftst1 - RX FIFO WCFG RAM\n",
            max_busy_cnt);
        fclose (fptr);
        return;
    }
    fprintf (fptr, "%.4x%.4x%.4x\n", read_reg(rftst4) & 0x3fff,
        read_reg(rftst3) & 0xffff,
        read_reg(rftst2) & 0xffff);
}
fprintf (fptr, "\n");

/* Dump RX FIFO read configuration RAM to file */
fprintf (fptr, "RX FIFO Read Configuration RAM\n");

```



```

for (ram_addr=0; ram_addr<256; ram_addr++)
{
    fprintf (fptr, "%.8x : ", ram_addr);
    write_reg(rftst1, (0x1000 + ram_addr) | 0x4000);
    busy_cnt = 0;
    while((read_reg(rftst1) & 0x8000) && (busy_cnt++ < max_busy_cnt));
    if (busy_cnt >= max_busy_cnt)
    {
        printf ("\n\nERROR: Max busy cnt of %d on rftst1 - RX FIFO RCFG RAM\n",
            max_busy_cnt);
        fclose (fptr);
        return;
    }
    fprintf (fptr, "%.1x%.4x%.4x\n", read_reg(rftst4) & 0x000f,
        read_reg(rftst3) & 0xffff,
        read_reg(rftst2) & 0xffff);
}
fprintf (fptr, "\n");

/* Dump RX FIFO data RAM to file */
fprintf (fptr, "RX FIFO Data RAM\n");
for (ram_addr=0; ram_addr<2048; ram_addr++)
{
    fprintf (fptr, "%.8x : ", ram_addr);
    write_reg(rftst1, (0x2000 + ram_addr) | 0x4000);
    busy_cnt = 0;
    while((read_reg(rftst1) & 0x8000) && (busy_cnt++ < max_busy_cnt));
    if (busy_cnt >= max_busy_cnt)
    {
        printf ("\n\nERROR: Max busy cnt of %d on rftst1 - RX FIFO DATA RAM\n",
            max_busy_cnt);
        fclose (fptr);
        return;
    }
    fprintf (fptr, "%.3x%.4x%.4x%.4x%.4x\n", read_reg(rftst6) & 0x0fff,
        read_reg(rftst5) & 0xffff,
        read_reg(rftst4) & 0xffff,
        read_reg(rftst3) & 0xffff,
        read_reg(rftst2) & 0xffff );
}
fprintf (fptr, "\n");

/* Close output file */
fclose (fptr);

```

```
/* take chip out of test mode */  
write_reg(0x0050, 0x0000);  
}
```

**Conclusion**

This application note enables the software to dump the DS31256 registers, queues, descriptors and FIFO RAM's to a specified file.

If you have further questions about our HDLC controller products, please contact the Telecommunication Applications support team via email [telecom.support@dalsemi.com](mailto:telecom.support@dalsemi.com) or call 972-371-6555.