

# **GC3355 Application NOTE**

**Version 1.0**

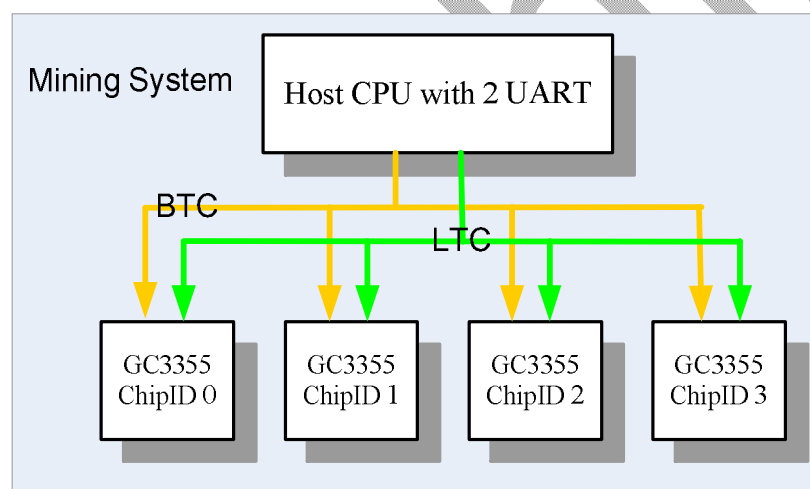
Version	Date	Description
1.0	2013-11-28	Initial version

GridChip

# 1. General Description

GC3355 supports two configuration mode: PN mode(CFG\_P/CFG\_N mode) and UART mode. PN mode is back compatible with previous BTC mining ASIC chips, and support BTC mining only. UART mode can support BTC and LTC, this Application note describe UART mode only.

The following is one example of Mining System, with one Host CPU and 4 GC3355 chips. The CPU will run two cgminer programs, one for BTC mining and another for LTC mining. Each GC3355 chip is assigned with one different ID to be accessed by the Host CPU. The ID is used as “chip address” in the Mining System.



The following is the communication command from Host CPU to GC3355:

55, AA, {CLUS\_ADDR[3:0], CHIP\_ADDR[3:0]}, REG\_ADDR[7:0], DATA[7:0], DATA[15:8], DATA[23:16], DATA[31:24]

The UART mode also supports burst command, the data can be 32\*N bits, for example:

55, AA, {CLUS\_ADDR[3:0], CHIP\_ADDR[3:0]}, REG\_ADDR[7:0], DATA0[7:0], DATA0[15:8], DATA0[23:16], DATA0[31:24], DATA1[7:0], DATA1[15:8], DATA1[23:16], DATA1[31:24], ..., DATAN-1[7:0], DATAN-1[15:8], DATAN-1[23:16], DATAN-1[31:24]

The DATA0, DATA1, ... DATAN-1 will be written to the following register address: REG\_ADDR, REG\_ADDR+1, ... REG\_ADDR+N-1.

Each GC3355 chip only responses the command with the chip\_id == the CHIP\_ADDR, or the CHIP\_ADDR is 0xf which denote broadcast. In another words, if the command is broadcast (the CHIP\_ADDR is 0xf), all the chips will response the command.

The CLUS\_ADDR denote the function module in the GC3355: 0 is BTC module, 1 is LTC module; 0xE is CPM module.

The REG\_ADDR is the register address in each function module. Please reference the register's description for detailed information.

## 2. BTC work Mode

The following code is an example: xclk is 25MHz, core clock is 500MHz, force\_start=0, rpt\_cycle = 0x10

```
cfg_chip_reg(8'hef, 8'h00, 32'h94e00005);
cfg_chip_reg(8'h0f, 8'hff, 32'h00000010);
cfg_btc_data; //configure the BTC's data, which address is 0x0—0xb
```

## 3. LTC work mode

the following code is an example: xclk is 25MHz, core clock is 500MHz, configure the UART's, timeout = 0x10

```
cfg_chip_reg(8'h1f, 8'h28, ltc_cfg_ctrl); // ltc_cfg_ctrl = 0x00000013
cfg_chip_reg(8'hef, 8'h30, ltc_cfg_reg); //ltc_cfg_reg = 0x00000020
cfg_chip_reg(8'hef, 8'h00, 32'h 94e00005); // pll config
cfg_chip_reg(8'hef, 8'h02, 32'h0000_0000); //gating BTC's core0- core31 clk
cfg_chip_reg(8'hef, 8'h03, 32'h0000_0000); //gating BTC's core31-core63 clk
cfg_chip_reg(8'hef, 8'h04, 32'h0000_0000); //gating BTC's core64-core95 clk
cfg_chip_reg(8'hef, 8'h05, 32'h0000_0000); //gating BTC's core96-core127 clk
cfg_chip_reg(8'hef, 8'h06, 32'h0000_0000); //gating BTC's core128-core159 clk
```

```
cfg_chip_reg(8'hef, 8'h20, uart_bps); // uart_bps
```

```
cfg_chip_reg(8'hef, 8'h21, uart_timeout); //uart_timeout= 0x80000018
```

```
cfg_ltc_data; //config the ltc's data, which address are 0x0—0x22, min/max nonce: 0x23, 0x24
```

## 4. BTC+LTC work mode

the following code is an example: xclk is 25MHz, LTC/BTC core clk is 500MHz, configure the UART's BPS

```
cfg_chip_reg(8'hef, 8'h20, uart_bps); // uart_bps
```

```
cfg_chip_reg(8'hef, 8'h21, uart_timeout); //uart_timeout= 0x80000018
```

```
cfg_chip_reg(8'hef, 8'h30, ltc_cfg_reg); //ltc_cfg_reg = 0x00000020
```

```
cfg_chip_reg(8'h1f, 8'h28, ltc_cfg_ctrl); // ltc_cfg_ctrl = 0x00000017
```

```
cfg_chip_reg(8'h0f, 8'hff, 32'h80000010); //btc's misc register
```

```
cfg_chip_reg(8'hef, 8'h00, 32'h94e00005); // pll config
```

```
fork
```

```
    cfg_btc_data
```

```
    cfg_ltc_data
```

```
join
```