

# Getting Started with the Winbond WTS701 TTS Chip

The WTS701 requires a minimum of external components to operate. The SPI interface allows connection to a microcontroller using between four and six pins. The examples here assume five pins connected adding the R/B\ (ready/not busy) pin to the standard SPI interface.

This guide explains how to send the necessary commands for the chip to say “Hello world” and provides help and troubleshooting if it does not say anything at all.

## Initial checklist

Before sending commands to the chip, run through the following list to ensure the chip is in a position to operate correctly. Check the following items:

- All pins are soldered to the board and no pins are shorted together.
- All No Connects are left floating
- Voltage supply pins 11, 12 and 48 are connected to 2.7 – 3.3 V
- The crystal is running at 24.576 MHz
- A 4.7  $\mu$ F capacitor is connected between pin 40 (ATTCAP) and ground
- The reset signal is LOW (pulsed HIGH for reset)
- The CS\ signal is LOW before sending commands to the WTS701
- Pins 13, 26 and 6 (INT\, R/B\ and MISO) pulled up to  $V_{CC}$
- Speaker outputs are differential and SP-/+ *not connected to ground*

## SPI Hardware interface

The SPI bus needs to be connected to a microcontroller or similar device. The following connections are needed:

### Outputs from microcontroller

SS\ - Slave select for starting an SPI command  
SCLK – Bit clock for SPI data  
MOSI – Data going in to the WTS701

### Inputs to microcontroller – Requires pull-up resistor to $V_{CC}$

MISO – Data from the WTS701  
R/B\ Ready to accept commands (optional but recommended)  
INT\ - Interrupt from WTS701 (optional)

## Implementation of SPI interface

When the conditions listed above are met, the first commands can be sent to the WTS701. For each byte of data sent to the WTS701, the chip will respond with a status byte so it is helpful to add the ability to read data at the same time as writing data. If writing code for a microcontroller, the following three functions are useful. The idle level of SCLK and MOSI can be either HIGH or LOW, but it is important that SS\ is HIGH at all times during startup so that the initialization of SCLK is not taken as a first clock.

## spi\_begin

When the SS\ line goes LOW, the WTS701 starts accepting incoming data on the SPI bus. The function should set SCLK and MOSI HIGH and put a HIGH to LOW transition on the SS\ pin. Now the chip is ready to accept data on the MOSI line.

```
void spi_begin (void)
{
    IO_PORT = MOSI_HI | SCLK_HI | SS_HI; // normal idle state
    IO_PORT = MOSI_HI | SCLK_HI | SS_LO; // SS goes low to start cmd
}
```

## SpiSendByte

Data is clocked into MOSI on the rising edge of SCLK. Output data on MISO from the WTS701 is valid while SCLK is HIGH. When sending a byte, data is clocked in MSB first. The function should follow this format:

1. Check the ready pin to ensure the chip is ready to accept data.
2. Before the rising edge on SCLK, set MOSI to the desired bit value; remember MSB is clocked in first.
3. Put a rising edge on SCLK.
4. Read the value on MISO, the bits are clocked out MSB first.
5. Put a falling edge on SCLK.
6. Repeat from step 2 until 8 bits are sent.
7. Return the byte read on MISO.

```
unsigned char SpiSendByte (unsigned char byteToSend)
{
    unsigned char i;
    unsigned char result = 0;

    /*
    ** The flow control using the R/B\ pin will wait until the chip **
    is ready to accept the command before attempting to send it.
    */
    while(!(IO_PORT & READY_PIN));

    i = 0x80; /* start sending MSB first */

    /*
    ** Loop through the byte, sending MSB first on the SPI interface
    ** could have used a for loop here, but the duty cycle is better
    ** if some processing is done before the rising edge of the clock
    */

    while(i)
    {
        if(byteToSend & i)
        {
            /* send 1, set data high before rising edge on SCLK */
            IO_PORT = MOSI_HI | SCLK_LO | SS_LO;
            i>>=1; /* done here for duty cycle symmetry */
            IO_PORT = MOSI_HI | SCLK_HI | SS_LO;
        }
        else
        {
            /* send 0, set data low before rising edge on SCLK */
            IO_PORT = MOSI_LO | SCLK_LO | SS_LO;
            i>>=1; /* done here for duty cycle symmetry */
            IO_PORT = MOSI_LO | SCLK_HI | SS_LO;
        }
    }
}
```

```

    }
    /* MISO pin will return a bit for each bit written in. */
    if(IO_PORT & MISO)
    {
        result += (i<<1);
        if(i==0)
            result++;
    }
    }
    return result;
}

```

## spi\_end

The WTS701 latches incoming data and executes a command when SS\ is taken HIGH. At this point, the WTS701 expects at least two bytes of data to have been received from the external controller.

```

void spi_end (void)
{
    IO_PORT = MOSI_HI | SCLK_LO | SS_LO; // levels at end of command
    IO_PORT = MOSI_HI | SCLK_LO | SS_HI; // SS goes high to finish cmd
}

```

## Command sequence for conversion and expected responses

This section details the commands to be sent to the chip for text conversion. The return bytes are valuable to verify correct operation and that the chip is in the correct state at all times.

Now it is time to send commands to the chip, assuming the WTS701 has  $V_{CC}$  applied before the first command. The command sequence will work from any state of the WTS701, but the return status might differ. Before sending the first command, make sure SS\ is HIGH so that the first command is accepted.

- 1. Start with sending the clock configuration command. This should always be sent before any other commands as it configures the clock to run with the correct crystal clock speed.**

```

spi_begin();
status0 = spi_send_byte(0x14); //CMD_CFG_CLK
status1 = spi_send_byte(0); //CFG_CLK_24M
SendSPIEnd ();

```

Expected response:

```

status 0 = 0x00
status 1 = 0x80

```

The chip is ready to accept commands but not powered up.

- 2. Send the power up command, now the chip is ready to accept commands**

```

spi_begin();
status0 = spi_send_byte(0x02); //POWER_UP
status1 = spi_send_byte(0);
spi_end();

```

```
Delay(0.001); // 1 ms power up delay
```

Expected response:

```
status 0 = 0x00  
status 1 = 0x80
```

The chip is ready to accept commands but not powered up. The status shows the status existing when the command is sent and does not reflect changes that the command itself would cause.

**3. Send a Read Interrupt command. This will clear any pending interrupts and return the status of the WTS701, which should now be ready for conversion.**

```
spi_begin();  
status0 = spi_send_byte(0x06); //READ_INTERRUPT  
status1 = spi_send_byte(0);  
data0 = spi_send_byte(0);  
data1 = spi_send_byte(0);  
spi_end();
```

Expected response:

```
status 0 = 0x05  
status 1 = 0x80  
data0 = 0x00  
data1 = 0x00
```

Now the status reflects a chip powered up, with the text buffer empty and ready to accept commands.

**4. Time to send the actual text data, the conversion command, the text and the end of text command will be sent in one batch and is detailed in the following sequence.**

```
spi_begin();  
status0 = spi_send_byte(0x81); //TTS_CONVERT  
status1 = spi_send_byte(0);  
status0 = spi_send_byte('h'); // send standard ascii codes  
status1 = spi_send_byte('e');  
status0 = spi_send_byte('l');  
status1 = spi_send_byte('l');  
status0 = spi_send_byte('o');  
status1 = spi_send_byte(' ');  
status0 = spi_send_byte('w');  
status1 = spi_send_byte('o');  
status0 = spi_send_byte('r');  
status1 = spi_send_byte('l');  
status0 = spi_send_byte('d');  
status1 = spi_send_byte(0x1a);  
spi_end();
```

If everything is going well, now the chip should be speaking the traditional greeting to the world.

Instead of the last byte with 0x1a (end of text character) we could have sent the spi\_end command immediately after the last 'd' in 'hello world' and then sent a FIN (0x4c) command to tell the chip to convert the text buffer.

## Troubleshooting

In the case that the chip remains silent after the text has been sent, the following tests can help identify the problem.

- 1. The WTS701 does not return any data on the MISO pin.**
  - Check  $V_{CC}$  pins for correct voltage.
  - Check the Reset pin and CS\ are both tied LOW.
  - Ensure MISO is pulled to  $V_{CC}$  through a resistor.
  - Check voltages on the SPI interface,  $V_{CC}$  for logic 1 and GND for logic 0.
  - Compare the SPI sequence to the waveforms to verify that MOSI data is valid on the rising edge of SCLK and MISO is polled while SCLK is HIGH. Check that SS\ is held LOW throughout the command period and goes HIGH at the end. Check for glitches and verify the setup times of all transitions against the data sheet.
  
- 2. The WTS701 returns data on MISO where the R/B\ bit seems fine, but the buffer indicator bits in Status Byte 0 are wrong (i.e. returns 0x00 instead of 0x05 after power-up)**
  - Ensure that the power-up sequence was sent correctly.
  - The R/B\ bit is hardware controlled, while the buffer indicators are controlled by the embedded microcontroller. The microcontroller relies on the crystal to be running. Check that the crystal provides the required 24.576 MHz clock and the crystal type and external components are of the values specified in the datasheet.
  - Send a Read Version (RVER, 0x12) four-byte command and read back the status in the last two bytes indicating hardware and firmware version. The values will vary depending on the chip, but should in any case, not be 0. If the firmware version is 0 and the crystal is determined to operate correctly, there is a microcontroller problem. Try updating the firmware. If that is not possible, replace the chip.
  
- 3. All status bits are correct, but the chip is not producing any speech.**
  - If using the speaker output, check that both SP- and SP+ are connected directly to the speaker terminals of a speaker with no less than 8 Ohm impedance. It is a differential signal and must not be grounded.
  - If using the AUX OUT signal, the output must be enabled first. Check the datasheet for setting up the AUD register.
  - Check that an end of text command has been sent, either by finishing the text with an end of text character, or by sending a FIN (0x4c) command after the convert command.
  
- 4. The chip outputs corrupted speech.**
  - Check that the characters sent are of the correct format (ASCII for English, Unicode or Big5 for Mandarin).
  - The firmware and corpus are not corresponding to the same version. Re-download firmware and corpus or swap the chip for a correctly programmed chip.

## SPI Waveforms

11-May-03

14:26:55

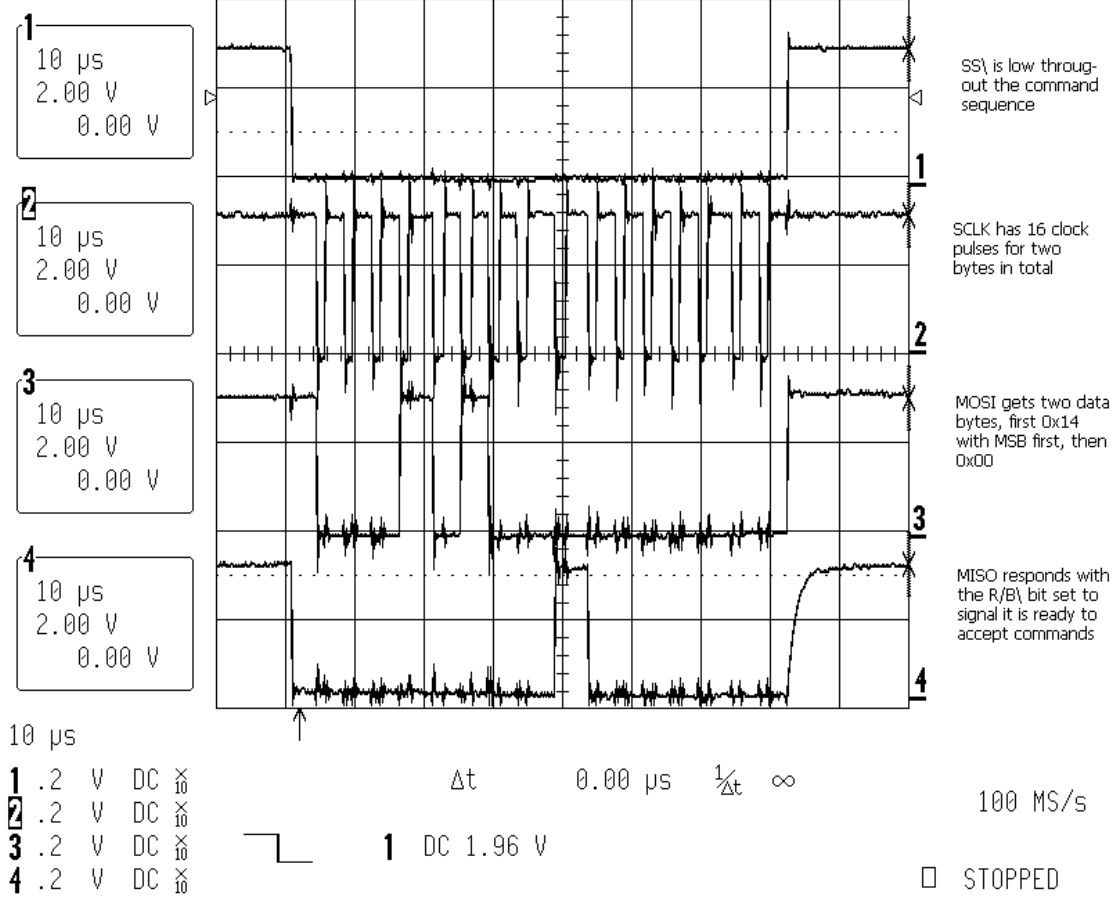


Figure 1 – Sending Clock Configuration command

11-May-03  
14:27:38

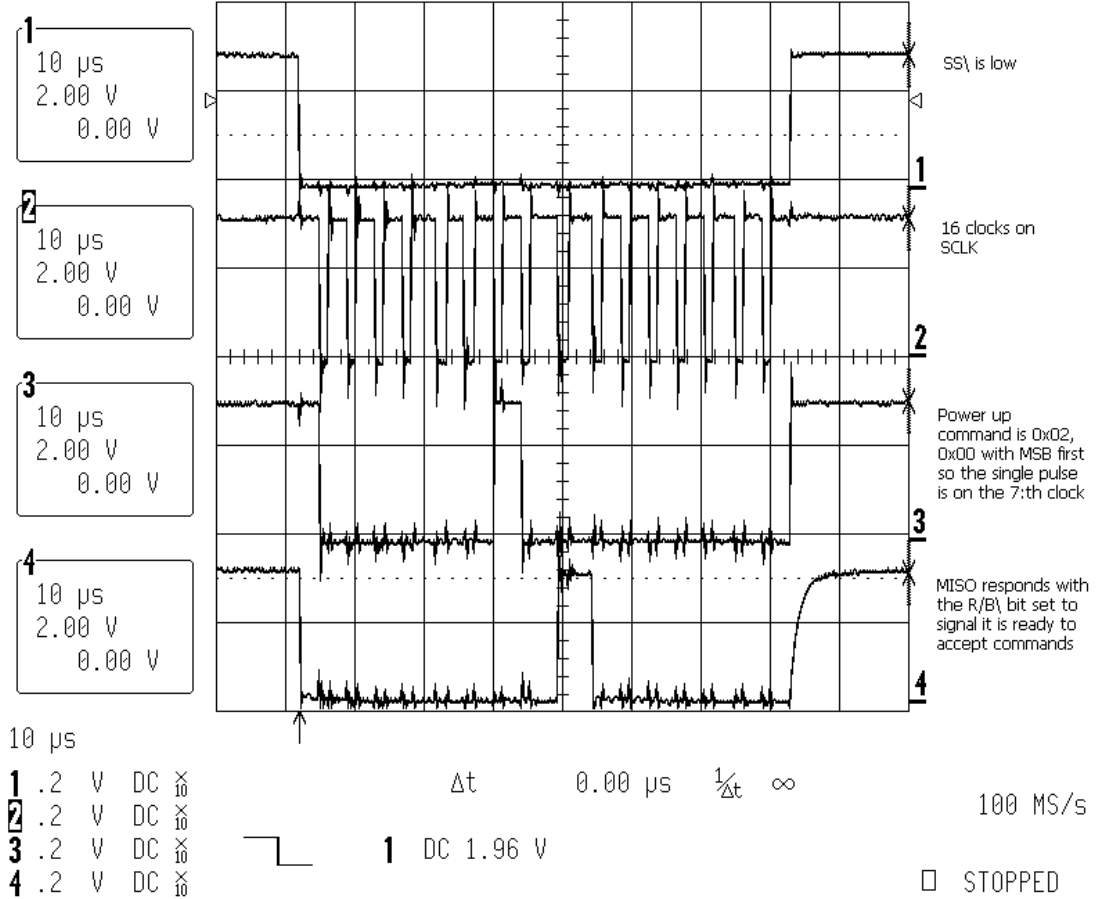


Figure 2 – Sending Power Up command

11-May-03  
14:28:25

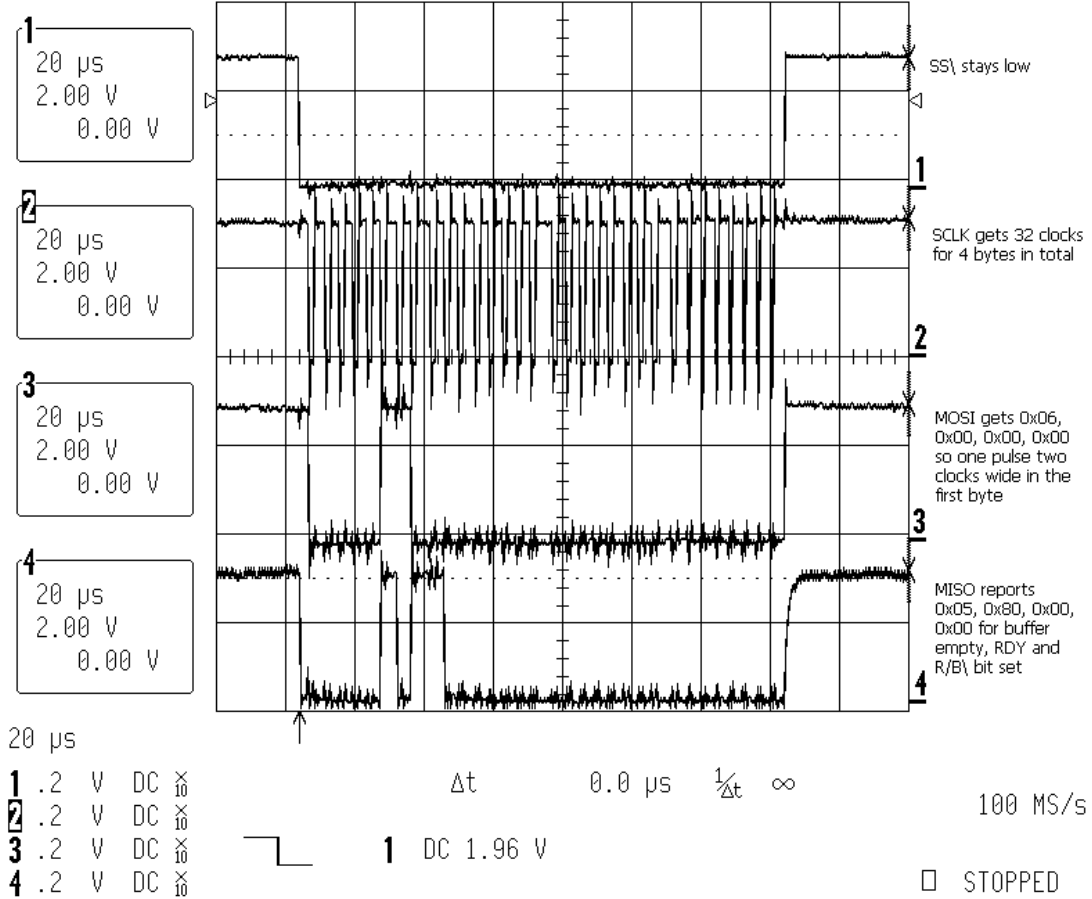


Figure 3 – Sending Read Interrupt command