

# 3WC Protocol

Craig Pataky  
November 20, 2008

## Introduction

The 3WC protocol is a 3-Wire signaling method designed for interconnecting microcontrollers of varying speed and capability to other microcontrollers. Unlike many interconnects -such as UART, I2C, SPI, CAN, and others- the 3WC protocol does not assume hardware assistance. Every processor capable of driving and reading port pins can utilize 3WC.

All 3WC compliant devices are either a master or a slave. For any given design, there can be only one master. There are no specific limits to the number of slaves.

## History

You know, there are a lot of protocols out there and you would be right to ask why we might want another one. Powerful processors now have so much interconnect built-in that communications work like magic as registers are loaded, unloaded, and interrupts are serviced. For \$10 I'll give you a competent processor indeed.

But hey, I live in a world dominated by \$2 processors.

And experience has taught me this:

Five \$2 processors have more computing  
power than one \$10 processor.

Here's an example: Just try to find a \$4 processor that has two UARTS. I can find lots of \$2 parts with one UART. So why not program one \$2 part to behave as a UART and make it a slave to another device?

While I'm at it, I can program that slave part to be smart enough to bother the master with data only if specific criteria are met. That's the power of parallel processing.

To make a long story short, 3WC was created to allow even the cheapest microcontrollers to efficiently talk to each other.

## Advantages

The 3WC bus requires only 3 pins.

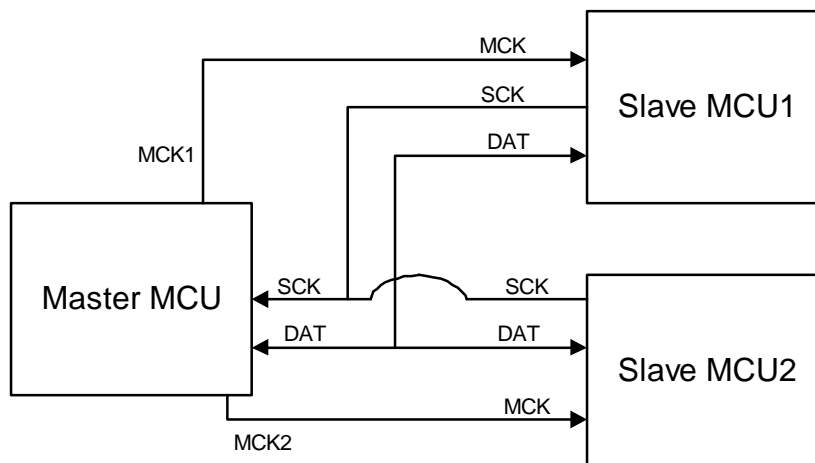
The 3WC protocol is not timing dependent. Every clock transition is acknowledged while each bit is transferred. You can effortlessly mix 20mhz devices with 1mhz devices.

Slave devices do not need to be immediately responsive to master devices. The master will request attention and the slave will provide attention when convenient.

Either the master or the slave can suspend a transfer to perform a more important activity, and resume the transfer without special consideration. This allows both the master and slave to service other interrupts while communicating.

Every transaction of the 3WC bus takes place as a single 10-bit data word transfer, MSB to LSB. Those bits can be defined arbitrarily. A typical use might be 2 function bits and 8 data bits.

## Example Wiring



**Figure 1: Slaves share SCK and DAT, but not MCK.**

Every slave device is given a dedicated master clock. The master clock serves double duty as both a clock source and chip select.

Most MCUs have provisions to enable weak pullups on port pins. If the processor does not have this capability, it is recommended that 100K pullups be placed on all signals associated with 3WC.

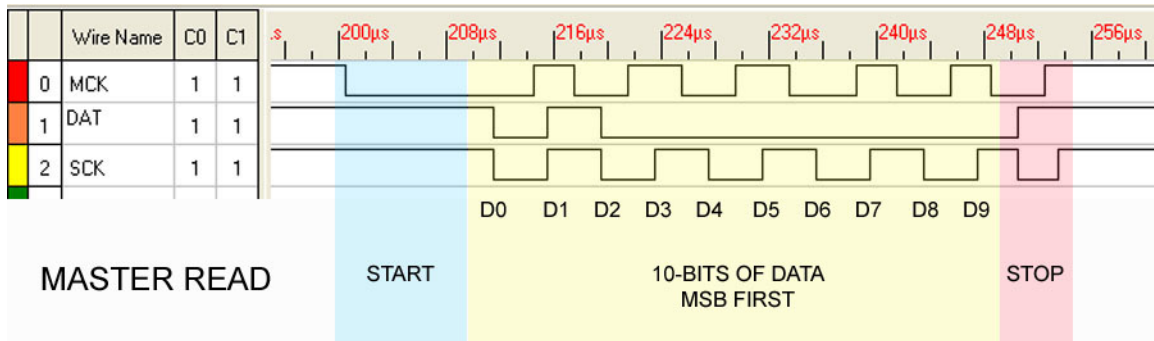
### 3WC Signals

MCK	Master Clock / Select	Driven by master during transfer, pulled up otherwise
DAT	Data	Driven during transfer, pulled up otherwise
SCK	Slave Clock	Driven by slave during transfer, pulled up otherwise

**Table 1: Only 3 signals are used in 3WC.**

### Master Read

To read data from a slave, the master floats DAT and drives MCK LOW. The slave drives DAT high or low depending on the bit to be transferred, and drives SCK LOW.



**Figure 2: The master reads the value 0x100 from the slave.**

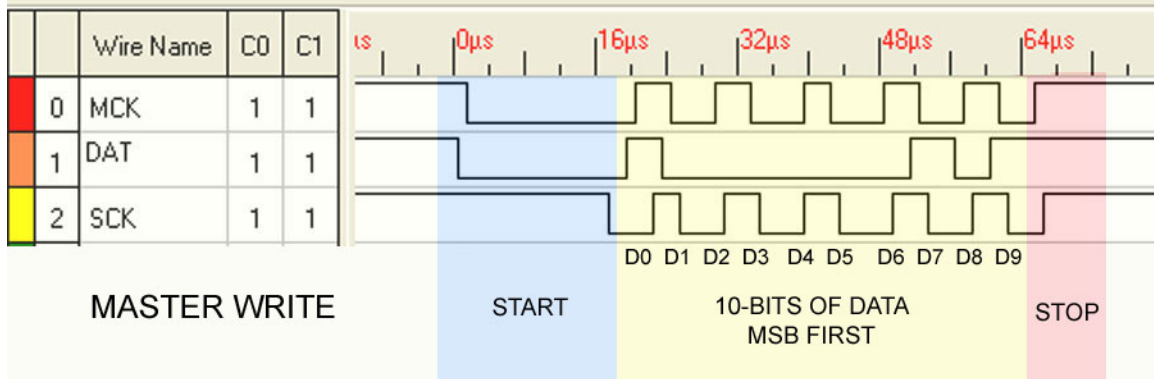
The master reads DAT, and sets MCK HIGH. The slave asserts the next bit on DAT and drives SCK HIGH. The master reads the bit and drives MCK LOW. This process is repeated for all ten bits.

When the master reads the last bit, MCK is driven LOW. The slave floats DAT and drives SCK LOW. When the master detects SCK LOW, the master drives MCK HIGH. The slave then drives SCK HIGH and the transaction is complete.

### Master Write

To write data to a slave, the master drives DAT LOW and then MCK LOW. The slave drives SCK low.

The master then asserts the data MSB on the DAT line and drives MCK HIGH. The slave reads the data and drives SCK HIGH. The master asserts the next bit on DAT and



**Figure 3: The master writes the value 0x205 to the slave.**

drives MCK LOW. The slave reads the data and drives SCK LOW. This process is repeated for all ten bits.

When the last bit has been read (slave drives SCK LOW) the master drives MCK HIGH. The slave drives SCK HIGH and the transaction is complete.

## Appendix A: Source Code

To illustrate the compact nature of the protocol in practice, I am providing source code for a master 3WC device here. This was created for an ATmega128 using the 2008 edition of GCC.

```
//Send data to the 3W bus as a master device .
BOOL MASTER3W_SendData(w )
{
    WORD j;

    setDAT(FALSE);
    setMCK(FALSE);
    while (getSCK()) asm volatile ("nop"); //Wait here until slave clock is low

    for (j=0;j<5;j++)
    {
        if (w&0x200) setDAT(TRUE); //Assert the data bit
        else setDAT(FALSE);
        w=w<<1;
        setMCK(TRUE); //Set the master clock HI
        while (!getSCK()) asm volatile ("nop"); //Wait for slave clock to go HI

        if (w&0x200) setDAT(TRUE); //Assert the data bit
        else setDAT(FALSE);
        w=w<<1;
        setMCK(FALSE); //Set the master clock LOW
        while (getSCK()) asm volatile ("nop"); //Wait for slave clock to go LOW
    }

    getDAT(); //Cleanup by ensuring data is an input
    setMCK(TRUE); //Set master clock to HI
    while (!getSCK()) asm volatile ("nop"); //Wait for slave clock to go HI
    return(TRUE); //Success!
}

//Send data to the 3W bus as a master device.
BOOL MASTER3W_GetData(WORD* pW)
{
    WORD j;
    WORD w;
    BOOL b;

    getDAT(); //Set data as input
    setMCK(FALSE); //Drive MCK low .
    w=0;

    for (j=0;j<5;j++)
    {
        while (getSCK()) asm volatile ("nop"); //Wait until slave clock goes LOW
        w=w<<1;
        b=getDAT(); //Read the data bit
        w|=b;
        setMCK(TRUE); //Set master clock HI

        while (!getSCK()) asm volatile ("nop"); //Wait for slave clock to go HI
        w=w<<1;
        b=getDAT();
        w|=b;
    }
}
```

```
    setMCK(FALSE);                //Set master clock LOW
}

while (getSCK()) asm volatile ("nop"); //Wait until slave clock goes LOW
setMCK(TRUE); //Set master clock to HI
while (!getSCK()) asm volatile ("nop"); //Wait until slave clock goes HI

*pW=w;
return(TRUE); //Success!
}
```

###