SILICON LABS

AN752

MEMS®

In-System Programming of the Si504 Device Using C1D Interface

1. Introduction

The Si504 device can be programmed via a single-pin, C1D interface. This application note relies on a small form-factor, stand-alone microcontroller (MCU), and code that accepts various commands to be executed on the Si504. The MCU firmware is partitioned to allow easy editing of the device configuration. Once created, the device configuration is simply downloaded to the MCU's flash memory.

Along with this application note, example code is provided that will be useful to users in implementing this on their system. This example code can be used as a reference to study the C1D communication and implement the same on any system.

2. Serial C1D Communication

2.1. Overview

The Si504 C1D interface uses Silicon Lab's patent-pending Transition Interval Code scheme, which uses pulse widths to determine logic states. The bit 0 has a nominal duration between 0.45 to 5.5 µs (T_{ZERO}) and the bit 1 has a nominal duration between 2.5 x (T_{ZERO}) to 16 µs (T_{ONE}).

The C1D interface supports a beginning C1D polarity of either high (C1D = 1) or low (C1D = 0). C1D steady state is provisioned as high; however, users can force C1D steady state low. Users should be aware the internal pull-up resistor will remain active. Given the above conditions, the diagrams below show both C1D polarity options.

Table 1shows the various ac characteristics of the C1D interface.

Table 1. Single Wire Interface AC Characteristics

Parameter	Symbol	Min	Typ	Max	Units
Bit "0" Nominal Duration	T_{ZERO}	0.45		5	µs
Bit "1" Nominal Duration	T_{ONE}	2.5 x T_{ZERO}		16	µs
Transaction Reset/ Abort Time	T_{RESET}	30	–	–	µs
Initial/Reset Sequence to 1st Command	T_{RSC}	1	–	–	ms
Sleep Wake Up Pulse Width	T_{WUP}	0.2	–	–	µS

2.2. Notes on the C1D Interface

1. Transactions start from the steady value of C1D. C1D can start with either C1D = 0 or C1D = 1 when there are no C1D transactions present. (Since the interface can start with either a 0 or a 1, all figures illustrate two waveforms, one for each steady state C1D value.)

Copyright © 2014 by Silicon Laboratories

2. Transactions end with the same C1D value as they started with.
3. The steady state C1D value can change at any time. Any steady state value change must be followed by a T_{RESET} time during which no C1D changes are allowed.
4. Bit 0 and bit 1 are valid if they conform to bit 0's and bit 1's nominal time duration found in Table 1. If bit 0 and bit 1 conform to Table 1, T_{ZERO} and T_{ONE} can change on a transaction by transaction basis.
5. The timing of bit 0 and bit 1 can vary from T_{ZERO} and T_{ONE} nominal duration by ±10%. If, for example, the shortest duration of bit 0 is 1 s, then the longest duration is 1 s / 0.9 x 1.1 = 1.22 s, and the nominal duration is 1 s / 0.9 = 1.1 s.
6. The minimum nominal duration of bit 1 is 2.5 times the nominal duration of bit 0. Bit 1's nominal duration tolerance is ±10%.
7. If the C1D value is held steady for T_{RESET} time in the middle of a transaction, the transaction will abort and all the data will be discarded. Figure 1 illustrates a C1D transaction abort.

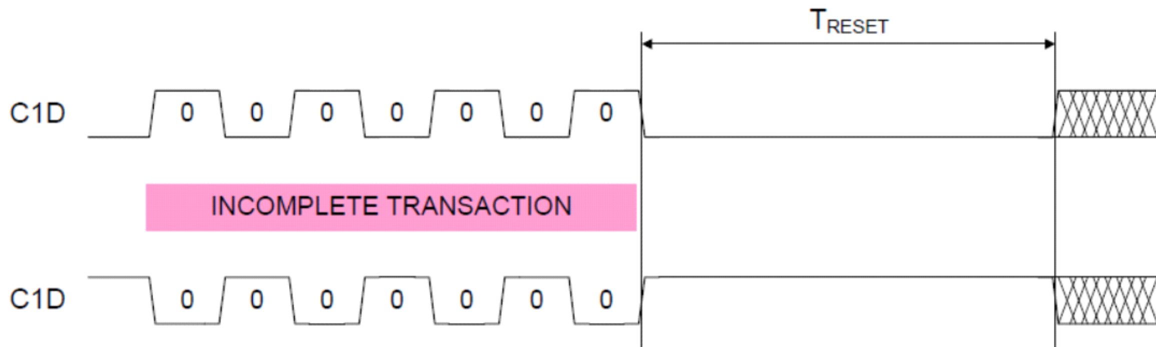

Figure 1. C1D Transition Abort

Figure 2 shows the C1D setup/reset sequence and the 0 and 1 transition timing.

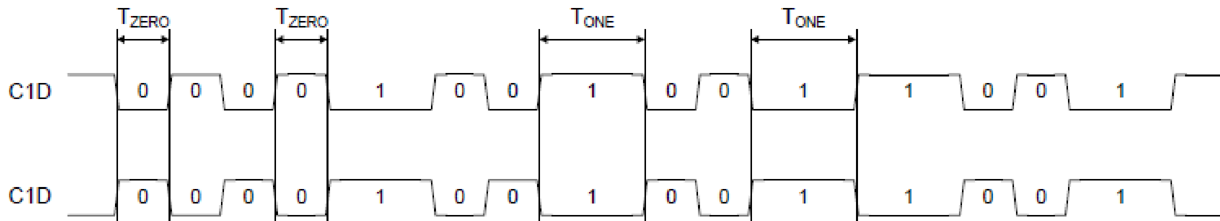

Figure 2. C1D Transition Setup/Reset and Interval Code

2.3. C1D Command/Byte Write Transaction/Issuing Commands

2.3.1. Command/Byte Write Instruction

The Command/Byte Write Transaction is used for the Command Byte and the Data Byte transfer during the command sequence. The Command/Byte Write Transaction is shown in Figure 2. (Note that the LSB bit D0 is transferred first during transactions.)

SILICON LABS

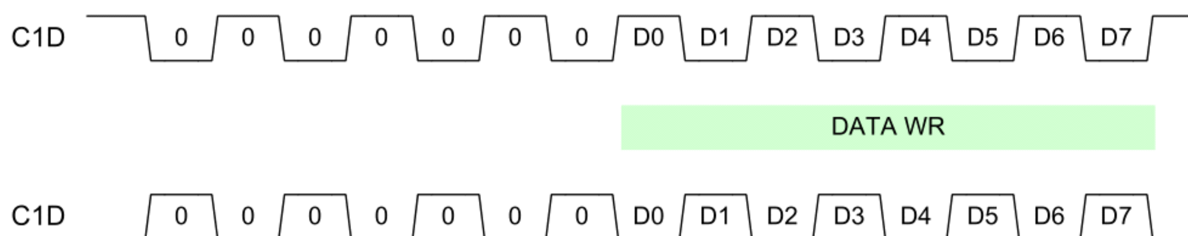

Figure 3. Data Write

2.3.2. Issuing Commands

1. Commands consist of one or more Commands/Byte Write Transactions.
2. Commands start with a Command Byte followed by 0 to 4 Data Bytes of the command argument.
3. Command and data write transactions within user commands must be separated by at least T_{CBI} time.
4. Commands are processed after the last argument data byte is sent.
5. Commands with data arguments can be reset and aborted by issuing a Setup/Reset Sequence in place of the data byte. Note that the transferred portion of the user command will be discarded.
6. Transactions must be separated by at least T_{CI} time.
7. After Setup/Reset Sequence, wait at least T_{RSC} time before issuing commands.
8. The command byte information and the command state diagram can be found in the Si504 data sheet.

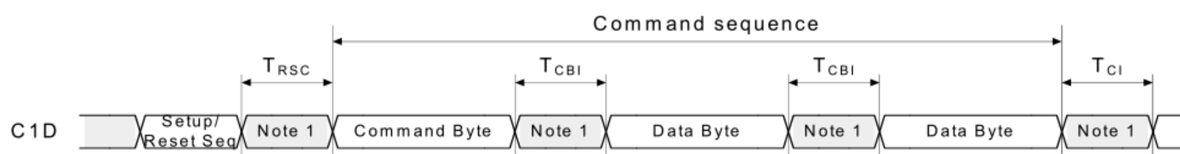

Figure 4. Command Sequence Timing

3. C1D Example Code for Si504

To help understand the C1D interface better, example code is provided which will assist the user in writing system-level code. This example code is written to support the Silicon Labs Si501-2-3-4-EVB using the Silicon Labs C8051F380 MCU; however, it is written in a manner to allow easy portability to other platforms.

Given below are some snippets of that code with some helpful comments. The AN752SW.zip package includes the example code.

3.1. Main Function

The main function in the example code contains calls to each of the Si504 commands as shown below. Each of these commands inherently calls the `si504_WriteCommand( )` function to transfer the command and data to the Si504.

```
//-----------------------------------------------------------------------------
//
//  The entry point of the application.  It sets up the hardware then waits forever
//
//-----------------------------------------------------------------------------
void main (void)
{

/*  Disable Watchdog Timer */
    PCA0MD &= ~0x40;

    Peripheral_Config();

/*  Send Commands to Si504 */

        //Specify frequency value in MHz.Correct values range from [0.032768-100MHz]
    si504_NewFrequency(32.768);
    /*  Delay here to see effect of the previous function call */

    si504_OffsetFrequency(0x1020);
    /*  Delay here to see effect of the previous function call */

    si504_Stop();
    /*  Delay here to see effect of the previous function call */

    si504_Doze();
    /*  Delay here to see effect of the previous function call */

    si504_Run();
    /*  Delay here to see effect of the previous function call */

    si504_Sleep();
    /*  Delay here to see effect of the previous function call */

    si504_Run();
    /*  Delay here to see effect of the previous function call */

        //specify the argument from the section Power/Jitter Configuration Options
        // section in si504.h
    si504_SetDriveStrength(SI504_DRV_STRENGTH_0p7ns_3p3V);
    /*  Delay here to see effect of the previous function call */

        //specify the argument from the section Drive Strength Configuration Options
        //section in si504.h
    si504_SetPowerJitterMode(SI504_PWR_JITTER_LOW_JITTER);
    /*  Delay here to see effect of the previous function call */

    si504_OrderedFrequency();

/*  Wait Forever  */
    while (1){}
}
```


SILICON LABS

3.2. Si504_WriteCommand()

The `c1_SendSetupReset` command is used to setup the C1 interface of the Si504 to receive commands.

```
//-----------------------------------------------------------------------------------------
//  si504_WriteCommand()
//-----------------------------------------------------------------------------------------
// This function sends commands to Si504 by first reseting si504 using c1_SendSetupReset()
// Then send the command with/without arguments(pointer data_bytes).Size of the arguments
// is defined as per the argument size_of_data
//-----------------------------------------------------------------------------------------
static void si504_WriteCommand(char command, char *data_bytes, int size_of_data)
{
    int i=0;

    c1_SendSetupReset();    //this is only needed after power-up
                            //and wakeup but doesn't hurt to do it every command

    c1_WriteData(command);

    for (i=0;i<=size_of_data;i++)
    {
        c1_WriteData(data_bytes[i]);
    }
}
```

3.3. c1_WriteData()

```
//-----------------------------------------------------------------------------------------
//  c1_WriteData()
//-----------------------------------------------------------------------------------------
//  Wrapper for the the code which actually writes data.The function below sets
//  the timer to provide necessary pulse width to 0 and 1.It converts the data from
//  hexadecimal to binary.
//-----------------------------------------------------------------------------------------
int c1_WriteData(unsigned char data8)
{   int error = C1_SUCCESS;
    int i;
    bit ea_saved;

/* Remember the EA status and disable interrupts */
    ea_saved = EA;
    EA=0;               // >= 2 byte instruction must follow

/*  Initialize TMR2 to count the pulse-time for a zero pulse  */
    c1_InitializeTimer(c1_zero_timer_count);


/* Convert the input data to bytes */
    for(i=0; i<8; i++)
    {   abSrl_Bits[i] = (data8>>i) & 0x1;
    }

/* Perform Data Write */
    srl_PerformDataTransaction();

/* Do not add extra 0 delay at the end .. it is at the beginning.
 * Disable TMR2 */
  TMR2CN &= ~(TMR2CN_TF2H_c | TMR2CN_TR2_c);

/* Restore interrupt enable status */
  EA = ea_saved;

  return error;
}
```

The entire srl_PerformDataTransaction() code is copied below with comments highlighted. Each step of the code follows the instructions shown by Figure 2.

```
//------------------------------------------------------------------------------------
//        srl_PerformDataTransaction()
//------------------------------------------------------------------------------------
//
//        Following code is a processor specific code and shows Fast Data Write instruction
//  using C1D.The code uses Timer2 to provide the necessary pulse width to the signal
//  the device 0 or 1.Pulse width for a zero is set using the timer to provide 3µs of
//  delay for 0 and the same is looped 3 times to get a delay of 3 x 3µs for a 1.
//  Make changes in the C1.h header file to modify OE pin,Dir Pin according to your
//        application
//------------------------------------------------------------------------------------

static void srl_PerformDataTransaction()
{

          /* -- Check whether the OE is properly driven .. DIR=0 required */
if (C1_Dir_Pin ==0)
          {
          /*  DIR=0 .. output. Make sure OE Pn.OE bit drive is push pull .. 1 */
                 C1_PnMDOUT |= C1_OE_Mask;

          /*  Get OE value */
                 gOeVal = (0 != C1_OE_Pin);

          /*  Enable TMR2 .. each transaction starts with 0 wait */
                 C1_TimerStart;

          /*  -- START */
          /*  Wait for TMR2 to overflow and clear the overflow flag. */
                 WaitForTimerOverFlowThenReset;

                 ToggleC1_OE_Pin;/* START .. first 0 begin */

             WaitForTimerOverFlowThenReset;

             ToggleC1_OE_Pin;       /* START .. second 0 begin */


             WaitForTimerOverFlowThenReset;

             ToggleC1_OE_Pin;     /* START .. third 0 begin */


             WaitForTimerOverFlowThenReset;

             ToggleC1_OE_Pin;     /* START .. fourth 0 .. begin */


             WaitForTimerOverFlowThenReset;

             ToggleC1_OE_Pin;     /* START .. fifth 0 begin */

             WaitForTimerOverFlowThenReset;

             ToggleC1_OE_Pin;     /* START .. sixth  0 begin */
```


SILICON LABS

```
        WaitForTimerOverFlowThenReset;

        ToggleC1_OE_Pin;    /* START .. seventh 0 begin */

    /*  -- DATA .. 8 bits */
        pbBit = abSrl_Bits;
        bBitNum = 8;
        do
        {

             /*    Wait for the tick */
          WaitForTimerOverFlowThenReset;

ToggleC1_OE_Pin;    /* DATA[m] .. start */

             /*    Decide how long to wait .. N-1 * Z if generating 1 */
          if ( 0 != *pbBit )
          {
            bLoop = bOneMultLess;
            do
            {
              WaitForTimerOverFlowThenReset;
            }
            while ( 0 != --bLoop );
          }
          pbBit++;
        }
        while ( 0 != --bBitNum );

    /*  Final D7 bit wait */
     WaitForTimerOverFlowThenReset;

    /*  Go back to the original value .. don't wait, done at higher level */
        ToggleC1_OE_Pin;

    }

}
```

3.4. Tips for Porting this Code onto Different Platforms

This code is written in such a way that it can be ported to other platforms by changing a few parameters.

1. The `Peripheral_Config()` function is used for initializing the Silicon Lab's C8051F380 MCU. It contains code to configure clocks, timers, interrupts, and I/O pins. This needs to be specific to the MCU/platform that you will use.
2. The `c1.h` file in the package contains MCU-specific pin definitions.

```
/*The following definitions are processor specific and should be changed to match your platform*/
#define     WaitForTimerOverFlowThenReset                   while(0==TF2H){}\
                                                            TF2H =0
sbit        P0_4                                     =      P0^4;
sbit        P0_0                                     =      P0^0;
#define     C1_IO_Port                                      P0
#define     C1_OE_Mask                                      0x01
#define     ToggleC1_OE_Pin                                 C1_IO_Port ^= C1_OE_Mask
#define     C1_PnMDOUT                                      P0MDOUT
#define     C1_OE_Pin                                       P0_0
#define     C1_Dir_Pin                                      P0_4
#define     C1_TimerStart                                   TR2=1
/*end of processor specific definitions*/
#endif  /* C1_HEADER  */
```

This file needs to change as per the platform being used.

3. Apart from these changes the rest of the code can be used as is for communicating with the Si504 device.

SILICON LABS

ClockBuilder Pro

One-click access to Timing tools, documentation, software, source code libraries & more. Available for Windows and iOS (CBGo only).

www.silabs.com/CBPro

Timing Portfolio
www.silabs.com/timing

SW/HW
www.silabs.com/CBPro

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

Disclaimer
Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

Trademark Information
Silicon Laboratories Inc., Silicon Laboratories, Silicon Labs, SiLabs and the Silicon Labs logo, CMEMS®, EFM, EFM32, EFR, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZMac®, EZRadio®, EZRadioPRO®, DSPLL®, ISOmodem ®, Precision32®, ProSLIC®, SiPHY®, USBXpress® and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com