# VHDL Model of the Dallas 1-Wire Digital Thermometer

## General Description

A VHDL model of the Dallas Semiconductor 1-Wire Digital Thermometer (DS1822) has been developed at the University of Missouri - Rolla. The model performs all the functions of the true DS1822 with the appropriate timing and gives feedback when errors occur.

Each VHDL DS1822 has a unique 64-bit serial code, set with a generic, which allows multiple slaves to function on the same 1-wire bus. Temperature is supplied as a real number through an additional input port.

## Applications

The VHDL model of the DS1822 is designed to allow simulation of hardware and software designed around the one-wire temperature device, so a design can be tested before it is implemented in hardware.

## Pin Configurations

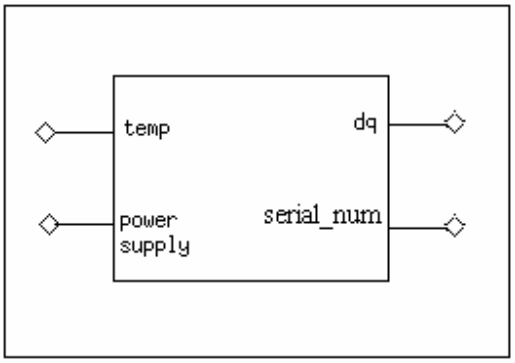



Figure 1. Pin configuration

## Features

- **Implements all ROM commands of DS1822 temperature sensor**
- **Implements all Function commands of DS1822 temperature sensor**
- **Unique serial number identifier can be set for each device**
- **Temperature can be varied using a type-real temperature input**

## Contact Information

Daryl Beetner
University of Missouri-Rolla
Department of Electrical and Computer Engineering
1870 Miner Circle
Rolla, MO 65409-1060

http://www.ece.umr.edu/courses/ee214

## Acknowledgements

Most of the timing diagrams shown in later figures were taken from the Dallas Semiconductor (now Maxim) DS1822 datasheet available at:

http://pdfserv.maxim-ic.com/arpdf/DS1822.pdf

## Pin Descriptions

| PIN Name | FUNCTION |
|---|---|
| dq | Data input/output pin. Also provides power to the device when used in parasite power mode. |
| temp | A real type generic input to provide the temperature read by the DS1822 during Convert T function command. |
| power_supply | A string type generic input to indicate the kind of power supply used by the DS1822. |
| serial_num | A bit_vector type generic input to provide the unique 64-bit serial code for each device. |

## Communication

### 1-Wire Protocol

Communication occurs via the Dallas 1-wire interface given in the datasheet of the DS1822. A brief description of the 1-wire protocol is provided here. Timing diagrams are shown in Figures 2-4.

### Bus Configuration

All 1-wire devices in a system may communicate and receive power over a single line named "dq". The 1-wire protocol uses a single bus master (micro-controller or microprocessor) to control one or more 1-wire slaves. The DS1822 is always a slave. One or more devices on the 1-wire bus can pull the bus low. All communication between the master and the slaves takes place using the bit timing supplied by the master.

### Reset and Presence Pulse

The bus master transmits a reset pulse by pulling the 1-wire bus low for between 480-960 μs. It then releases the bus and goes into receive mode. The DS1822 waits for 15-60 μs and then transmits a presence pulse by pulling the 1-wire bus low for 60-240 μs.

### Time Slots

Communication of individual bits is achieved through the use of "time slots", which allow the data to be transmitted over the 1-wire bus. Write slots are used by the bus master to write data to the slaves (DS1822s). Read slots are started by the master and allow the slaves to write information back to the master. Both read and write time slots are a minimum of 60 μs in duration with a minimum of 1 μs recovery time between two consecutive slots. One bit of data is transmitted over the 1-wire bus per time slot. Any data transmitted on the 1-wire bus starts with the least-significant-bit first.

### Write Time Slots

There are two types of write slots: Write 0 and Write 1 time slots. The bus master uses a Write 0 time slot to write a logic 0 to the DS1822 and Write 1 time slot to write a logic 1. The master writes a 1 by pulling the 1-wire bus low, then releasing it within 15 μs, and then lets it stay high for at least 45 μs. To write a 0, the bus

master pulls the line low for 60-120 µs. The DS1822 samples the 1-wire bus between 15 and 60 µs after the start of the time slot. If the bus is high, the DS1822 assumes a 1 has been written, otherwise it assumes a 0.

### Read Time Slots

The master initiates the transfer of each bit from the DS1822 using read time slots. The master begins a read time slot by pulling the 1-wire bus low for a minimum of 1 µs and then releasing it. The DS1822 can transmit a 1 back to the master by letting the bus go back high, or can transmit a 0 by keeping the bus low for a minimum of 15 µs. The bus master samples the bus state within 15 µs from the start of the time slot.

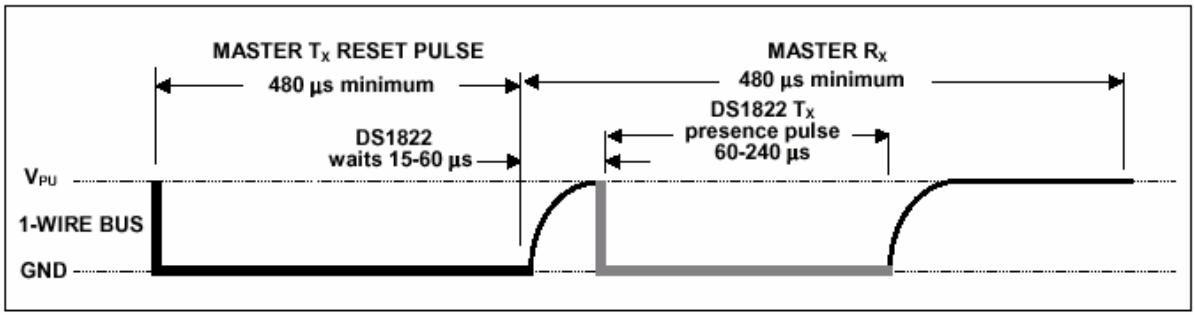



Figure 2. Reset and Presence pulse (From the DS1822 datasheet).

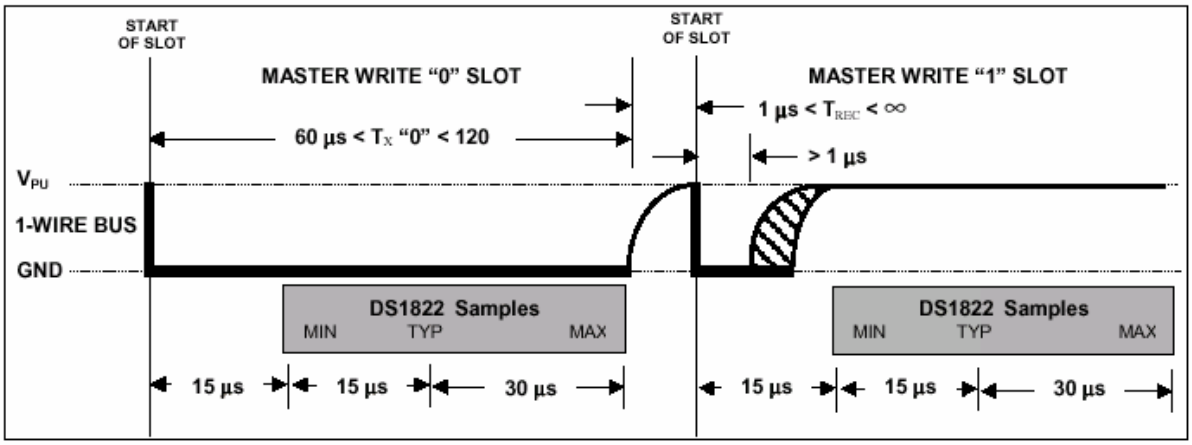
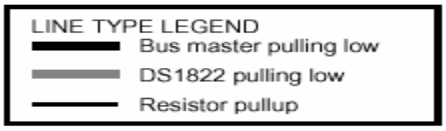



Figure 3. Write 0 and Write 1 time slot (From the DS1822 datasheet).



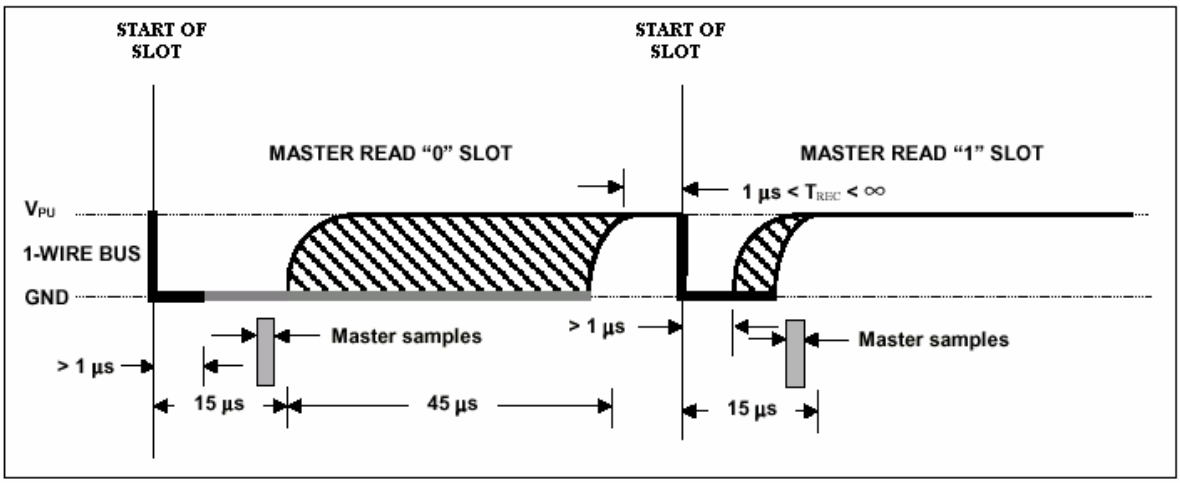
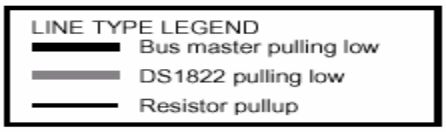



Figure 4. Read 0 and Read 1 time slot (From the DS1822 datasheet).




## Deviations From the True Hardware

Deviations of the VHDL model from the actual DS1822 hardware are as follows:

1. Each DS1822 has a unique 64-bit serial code stored in an on-board ROM. In the VHDL model of DS1822, a generic signal "*serial_num"* is defined with the architecture, which holds the 64-bit serial code.
2. Generic *power_supply* indicates the kind of power supply (*external or parasite*) used by the VHDL model of the DS1822.
3. During the Convert T command, the DS1822 temperature sensor will sense the temperature in its vicinity and convert it into a digital value. In the VHDL model of the DS1822, an input "*temp"* of type real is provided, which supplies the value of the ambient temperature.

## Model Operation

The VHDL model is constructed as a simple state machine, as shown in Figures 5 and 6.

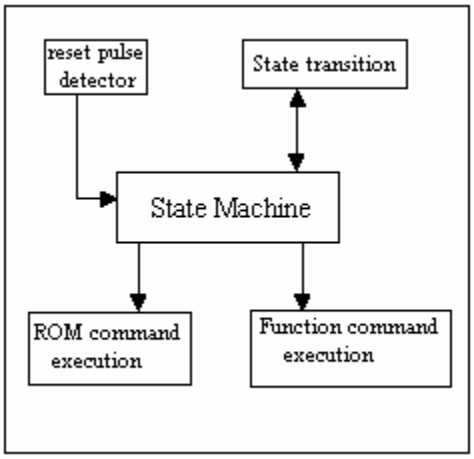



Figure 5. Block diagram of DS1822 model.

The state machine has the following four high-level states:

1. Init state
2. Read state
3. Romcomm state
4. Funccomm state

Each of these high-level states may contain lower-order sub-states. The default state is *init*. In the *init* state, the slave waits for the master to send a reset pulse. If a reset pulse is detected, the model waits for 15-60 microseconds and then transmits a presence pulse and enters the *read* state. If a reset pulse during any operation, the state machine will go back to the *init* state, send a presence pulse, and re-enter the *read* state.

Values are read (via write-0 and write-1 time slots) during the *read* state. The *read* state is encountered twice during a complete command sequence. During each encounter, the DS1822 model receives an 8-bit command from the master. These bits are read and placed in the command register. When the state machine enters the *read* state for the first time, it receives a ROM command from the master and enters the *romcomm* state. When it enters the *read* state for the second time after a reset pulse, it receives a function command from the master and enters the *funccomm* state.

In the *romcomm* state, the DS1822 model executes the ROM command received in the previous *read* state. Each ROM command is implemented in a different process. If the command is invalid, the state machine will re-enter the *init* state. Otherwise, after execution of the ROM command, the state machine will re-enter the *read* state so it can receive a function command from the master.

The state machine will enter the *funccomm* state after completing the *read* state for the second time, i.e. after completing the *romcomm* state and reading in a function command. The function command received from the master is executed in the *funccomm* state. As in the *romcomm* state, each function command is implemented in a different process and if the command is invalid the model will re-enter the *init* state. After completing a valid function command, the model will also re-enter the *init* state.

State transitions are controlled internally using control signals. For example, when a reset pulse is detected, the *reset* control signal is asserted, telling the model to enter the *init* state. Similar control signals are generated at the end of ROM and function commands. The *state_trans* process helps decide the next state of the state machine.

Further details are available in the VHDL code itself.

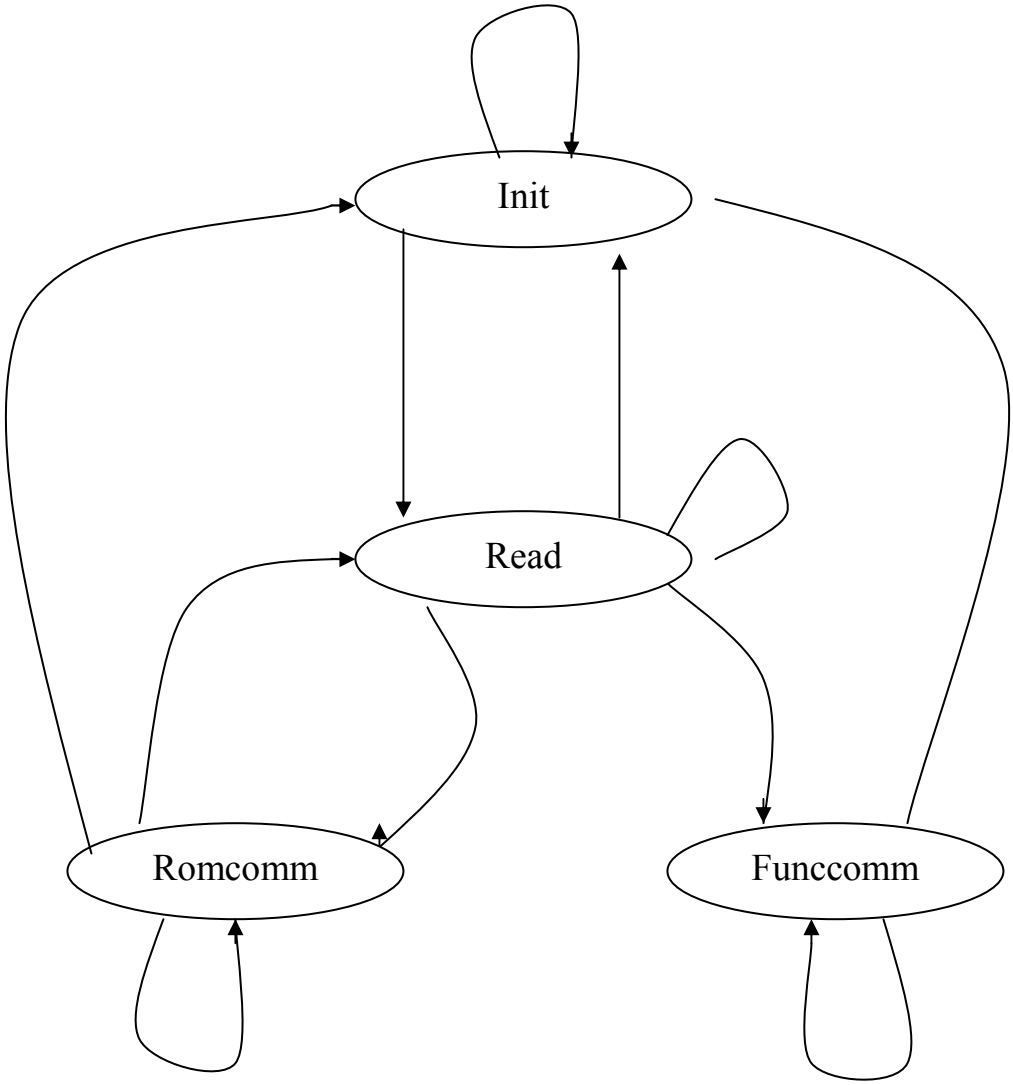



Figure 6. High-level model of state-machine.

## 8051 Interfacing

Below is a sample program to interface the VHDL model of DS1822 with an 8051 in a mentor graphics environment.

```
/* This program performs the following tasks:
        1. Issues a reset pulse
        2. Checks for the presence pulse
        3. Issues a Skip ROM command
        4. Issues a Read Scratchpad command
        5. Provide read time slots to read first two bytes of scratchpad (byte 0 and byte 1) */

#include <reg51.h>
#include <stdio.h>
#define uchar unsigned char
#define uint unsigned int
#define CLKPERIC 12          // 12 for regular 8051, 6 for 89C51Rx2's
#define FCLK 12              //clock freq in Hz
#define SKIPROM 0xCC
#define READSP 0xBE
sbit OWDQ= P1^5;             /* one wire DQ line */

bit reset_1wire(void);
void msec(uint t);
void U60(void);
void OW_writebit(bit b);
bit OW_readbit(void);
void OW_writebyte(uchar c);
uchar OW_readbyte(void);
uchar TempLSB;
uchar TempMSB;

void main(void){
 bit ow_present;
 int temp;
 int tempF;
 init_uart();
 while(1){
  ow_present=reset_1wire();
        if (ow_present) {                       // a presence pulse given
                OW_writebyte(SKIPROM);
                OW_writebyte(READSP);
```

**Sample 8051 Source (continued)**

```
                TempLSB=OW_readbyte();     // storing the byte 0 of scratchpad in TempLSB
                TempMSB=OW_readbyte();   // storing the byte 1 of scratchpad in TempMSB
        }
        msec(1000);
        };
 }  // end of main function

/* delay for t msec.  Use timer 0 */
void msec(uint t){
#define T1000 (-1000+22)*FCLK/CLKPERIC
 TMOD=(TMOD&0xF0) | 0x01;                    /* 16bit timer mode */
 while (t>0) {                               /*delay 1 msec */
  TH0= (T1000) >> 8;                         /* upper half of -1000 (0xfc) */
  TL0= (T1000) & 0xff;                       /* lower half of -1000 (0x18) */
  TR0= 1;                                    /* start timer 0 */
  while (~TF0);                              /* wait for TF0=1 */
  TR0= 0;                                    /* stop timer and clear overflow bit */
  TF0= 0;
  t=t-1;
 }
}

/* use Timer 0 to delay 480 uSec.
*/
void U480(void){
#define T480 (-480+10)*FCLK/CLKPERIC
  TMOD=(TMOD&0xF0) | 0x01;                  /* 16bit timer mode */
  TH0= (T480) >> 8;                         /* upper half of -480 */
  TL0= (T480) & 0xff;                       /* lower half of -480 */
  TF0=0;                                    /* make sure TF0 is clear */
  TR0=1;                                    /* start timer 0 and return*/
  }

/* use Timer 0 to delay 60 uSec.
*/
void U60(void){
#define T60 (-60)*FCLK/CLKPERIC
  TMOD=(TMOD&0xF0) | 0x01;                  /* 16bit timer mode */
  TH0= (T60) >> 8;                          /* upper half of -60 */
  TL0= (T60) & 0xff;                        /* lower half of -60 */
  TF0=0;                                    /* make sure TF0 is clear */
  TR0=1;                                    /* start timer 0 and return*/
  }
```

**Sample 8051 Source (continued)**

```
/* Function to write a bit */

void OW_writebit(bit b){
 U60();                          /* start 60 us clock */
 OWDQ=0;                         /* generate clock edge */
 OWDQ=b;                         /* write bit */
 while(!TF0);
 OWDQ=1;
 TR0=0;
 return;
}

/* Function to write a byte */

void OW_writebyte(uchar c){
 uchar i;
 for(i=8; i>0;i--){
  OW_writebit(c&1);
  c=c>>1;
  };
 return;
}


/* Function to read a bit from DS1822 */

bit OW_readbit(void){
 bit rc;
 uchar t;
#define WAIT10US 10;
 U60();
 OWDQ=0;
 t=WAIT10US;
 OWDQ=1;
 while(t>0) t--;                 /*spin wait loop about 10 us */
 rc=OWDQ;                        /* sample DQ */
 while(!TF0);
 TR0=0;
 return(rc);
 }
```

**Sample 8051 Source (continued)**

```
/* Function to read a byte from DS1822 */

uchar OW_readbyte(void){
 uchar rc=0;
 uchar i;
 for(i=8;i>0;i--){
  rc=(rc>>1)|((uchar)OW_readbit()<<7);
  };
 return(rc);
}



/* Generate a 480+ us low pulse on OWDQ port, and
   wait for 480 us for presence pulse from ds1822
   return 1 if presence else 0.  Uses T0 for timing
*/
bit reset_1wire(void){
 bit rc;
 /* hold DQ low for 480 uSec */
 OWDQ=0;
 U480();
 while(~TF0);
 TR0=0;
 TF0=0;
 OWDQ=1;
 /* wait up to 480 usec for DQ low (presence pulse) */
 U480();
 rc=0;
 while(~TF0) if (~OWDQ) rc=1;  /* a 1-wire device found! */
 TR0=0;
 return(rc); /* timeout waiting for presence pulse */
 }
```