

# Digilent Adept Two Wire Serial Interface (DTWI) Programmer's Reference Manual

Revision: August 16, 2010



1300 NE Henley Court, Suite 3  
Pullman, WA 99163  
(509) 334 6306 Voice | (509) 334 6300 Fax

## Introduction

This document describes programming interface to the Digilent Adept Two Wire Serial Interface (DTWI) subsystem for version 2 of the Digilent Adept software system. It describes the capabilities of the DTWI subsystem and the API functions used to access its features.

The DTWI subsystem provides access to a two wire serial interface but (TWI) that is compatible with the Inter-Integrated Circuit (I2C) bus developed by Philips semiconductor. The TWI bus is a low bandwidth, multi-point synchronous serial bus that uses two open drain, bi-directional signal lines: SCL (clock) and SDA (data). Devices on the bus can be either master devices or slave devices. Master devices initiate and terminate transactions on the bus. The master will generate the clock signal during a transaction. Either the master or the slave can drive the data line during a transaction depending on the nature of the transaction. During a transaction, a slave can cause the master to wait by holding the clock low.

The TWI bus uses a 7-bit address allowing up to 128 addressable devices. Device addresses are in the range 0-127, with address 0 being reserved as a broadcast address. When the address is placed on the bus during a transaction, the address is presented in the high order bits of a byte. The device documentation for some I2C devices document the address as being the seven bit address shifted left by one bit position.

Transaction on the TWI bus are made up of sequences of the following events:

**START condition:** A START condition is used to begin a bus transaction when the bus is in the idle state.

**STOP condition:** A STOP condition is used to end a transaction and return the bus to the idle condition.

**REPEATED START:** This signals the beginning of a new transaction immediately following a previous transaction without allowing the bus to return to the idle state.

**Address packet:** An address packet is made up of a slave address and a read/write indicator. All transactions must have an address packet, which indicates the slave device and the direction of data flow. Write transaction send data from the master to the slave. Read transaction send data from the slave to the master.

**Data packet:** All data packets are nine bits long and are made up of an 8-bit data byte followed by an acknowledge bit.

Transactions on the TWI bus are initiated by a bus master and addressed to a slave. A transaction can be begun when the bus is idle. A master begins a transaction by asserting a START condition on the bus. Following the start condition, the master sends an address packet to indicate the slave

address of the intended recipient of the transaction. An address packet is made up of the slave address followed by a READ or WRITE indicator bit (SLA+R or SLA+W). During a READ transaction the data will be sent from the slave to the master. During a WRITE transaction, data will be sent from the master to the slave.

After the address packet, the addressed slave will indicate that it is able to accept the transaction by asserting an ACK on the bus. If the slave is unable to accept the transaction, it will assert a NAK on the bus.

Following the address packet, the transmitter for the transaction will place data packets on the bus.

Once the required number of data packets, the master can end the transaction and release the bus by asserting a STOP condition. Alternatively, the master can begin a new transaction without release the bus by asserting a REPEATED START condition.

All DTWI API calls return a Boolean value: TRUE if the call is successful, FALSE if not successful.

## Port Properties

The following port property bits are defined for this subsystem:

1. `dprpTwiMaster` – device supports TWI master capabilities
2. `dprpTwiSlave` – device supports TWI slave capabilities
3. `dprpTwiMultiMaster` – device supports multi-master arbitration
4. `dprpTwiBatch` – device provides TWI batch support
5. `dprpTwiSetSpeed` – device has settable clock speed
6. `dprpTwiSmbAlert` – device supports SMB Alert pin
7. `dprpTwiSmbSuspend` – device supports SMB Suspend pin
8. `dprpTwiSmbPEC` – device supports SMB packet error checking

## DTWI Master Operation

The ability for a port to operate as a DTWI master is indicated by the `dprpTwiMaster` port property. The DTWI master functions can only be used with ports that support this property.

The DTWI bus is capable of supporting multiple masters on the same bus with priority arbitration if multiple masters attempt to access the bus simultaneously. DTWI ports that support multi-master operation are indicated by the `dprpDtwiMultiMaster` port property. A DTWI master that doesn't support this property must be the only master on the bus.

When operating as a bus master, the Digilent Adept compatible device will initiate bus transactions, write data to slave devices, and read data from slave devices in response to calls to the DTWI master functions.

## DTWI Slave Operation

The ability for a port to operate as a DTWI slave device is indicated by the `dprpTwiSlave` port property. The DTWI slave functions can only be used with ports that support this property.

When operating as a slave device, the Digilent Adept compatible device will accept bus transactions directed at its address. Data can be buffered by the device to be read from the device by a bus master. Data written to the device by a bus master will be buffered until read by the host. The Digilent Adept compatible device will have a limited amount of buffer memory available for buffering the data to be read from it and written to it by bus masters.

The port must be enabled for slave operation by calling `DtwiSlaveEnable`. This function assigns the slave device address as well as enabling operation as a slave. Data to be buffered by the device to be read by a bus master can be sent to the device before `DtwiSlaveEnable` is called.

Data to be read by a bus master is sent to the device using the function `DtwiSlaveTxPost`. The `DtwiSlaveTxQuery` function can be used to query the amount of data remaining in the transmit buffer.

Data written to the device by a bus master is stored in the device until read using the `DtwiSlaveRxRead` function. The `DtwiSlaveRxQuery` function can be used to determine the amount of data currently buffered by the device.

## DTWI Master Batch Operation

Master batch mode operation allows scripting a sequence of DTWI bus transactions. A script of batch commands specifying the bus transactions to be performed is placed in a buffer. The batch buffer is sent via the `DtwiMasterBatch` function call for execution. Support for batch mode operation is indicated by the `dprpTwiBatch` port property. Ports without this property do not support batch mode operation.

The batch command buffer is filled with a sequence of batch commands and their parameters. Data to be sent to slave devices is specified as parameter data to commands and is therefore part of the command buffer. Any data read from slave devices on the bus is returned in a separate data buffer. If multiple reads are specified by the batch command buffer, the data returned by each read is packed in the return data buffer.

The batch commands and their parameters are byte oriented binary data. The commands and their parameters are placed in the command buffer as a packed sequence of bytes. The symbols related to batch commands are defined in the header file `dtwi.h`. Eight batch commands are defined.

1. `tcbStop` – assert a stop condition on the bus.  
Parameters bytes: None
2. `tcbStartSlaw` – assert a start condition on the bus for write  
Parameter bytes:
  - a. Slave address
3. `tcbStartSlar` – assert a start condition on the bus for read  
Parameter bytes:
  - a. Slave address
4. `tcbRepStartSlaw` – assert a repeated start condition on the bus for write  
Parameter bytes:
  - a. Slave address

5. tcbRepStartSlar – assert a repeated start condition on the bus for read  
Parameter bytes:
  - a. Slave address
  
6. tcbPut – perform a master put (write) transaction on the bus  
Parameter bytes:
  - a. Low byte of count of bytes to Put (actual count value is a WORD)
  - b. High byte of count of bytes to Put
  - c. First data byte to put
  - d. Second data byte to put
  - e. ....
  
7. tcbGet – perform a master get (read) transaction on the bus  
Parameter bytes:
  - a. Low byte of count of bytes to Get (actual count value is a WORD)
  - b. High byte of count of bytes to Get
  
8. tcbWait – wait the specified number of microseconds  
Parameter bytes:
  - a. Low byte of wait period in micro seconds (actual value is a WORD)
  - b. High byte of wait period in micro seconds

## DTWI API Functions

The following API functions make up the TWI interface.

### **DtwiGetVersion(char \* szVersion)**

*Parameters*

szVersion - pointer to buffer to receive version string

This function returns a version number string identifying the version number of the DTWI DLL. The symbol `cchVersionMax` declared in `dpcdecl.h` defines the longest string that can be returned in `szVersion`.

### **DtwiGetPortCount(HIF hif, INT32 \* pcprt)**

*Parameters*

hif - open interface handle on the device  
pcprt - pointer to variable to receive count of ports

This function returns the number of TWI ports supported by the device specified by `hif`.

### **DtwiGetPortProperties(HIF hif, INT32 prtReq, DWORD \* pdprp)**

*Parameters*

hif - open interface handle on the device  
prtReq - port number to query  
pdprp - pointer to variable to return port property bits

This function returns the port properties bits for the specified TWI port. The port properties bits indicate the specific features of the DTWI specification implemented by the specified port.

### **DtwiEnable(HIF hif)**

*Parameters*

hif - open interface handle on the device

This function is used to enable the default TWI port (port 0) on the specified device. This function must be called before any functions that operate on the TWI port may be called for the specified device.

### **DtwiEnableEx(HIF hif, INT32 prtReq)**

*Parameters*

hif - open interface handle on the device  
prtReq - TWI port number

This function is used to enable a specific port on devices that support multiple TWI ports. This function must be called before any functions that operate on the TWI port may be called. The `prtReq` parameter specifies the port number of the TWI port to enable.

**DtwiDisable(HIF hif)***Parameters*

hif - open interface handle on the device

This function is used to disable and end access to the currently enabled TWI port on the specified interface handle.

**DtwiSetSpeed(HIF hif, DWORD freqReq, DWORD \* pfreqSet)***Parameters*

hif - open interface handle on the device  
freqReq - value of frequency (in Hz) to clock SCL signal at  
pfreqSet - pointer to return actual set SCL frequency

This function is used to specify the nominal bus clock speed used when the device functions as a TWI master. The device will select the highest speed it is capable of that doesn't exceed the requested speed and the actual speed selected will be returned. If the requested speed is less than the lowest speed supported by the device, the lowest supported speed will be used.

The default bus clock speed will be the highest clock speed that the device is capable of that does not exceed 400Khz.

**DtwiGetSpeed(HIF hif, DWORD \* pfreqReq)***Parameters*

hif - open interface handle on the device  
pfreqReq - pointer to return current SCL frequency (in Hz)

This function returns the current bus clock speed used when the enabled TWI port functions as a bus master. This function should only be called for ports that support settable bus clock speeds as indicated by the port properties bit for the enabled port.

## DTWI Master API Functions

### **DtwiMasterPut(HIF hif, BYTE dadr, DWORD cbSnd, BYTE \* rgbSnd, BOOL fOverlap)**

#### *Parameters*

hif	- open interface handle on the device
dadr	- TWI device address to slave to receive the data
cbSnd	- number of bytes to send
rgbSnd	- buffer of data to send to the slave
fOverlap	- TRUE for overlapped operation

This function is used to send a packet of data bytes to the specified TWI slave on the TWI bus associated with the port currently enabled on the specified interface handle. The TWI master will assert a START condition on the bus. If this succeeds, it will send the slave address plus write (SLA+W), transmit the specified number of data bytes, and then assert a STOP condition on the bus.

If the slave fails to ACK either the address or any of the data bytes, the transaction will be halted, a STOP condition asserted on the bus, and an error code returned indicating what error occurred.

### **DtwiMasterGet(HIF hif, BYTE dadr, DWORD cbRcv, BYTE \* rgbRcv, BOOL fOverlap)**

#### *Parameters*

hif	- open interface handle on the device
dadr	- device address of TWI slave to read from
cbRcv	- number of bytes to read from slave device
rgbRcv	- buffer to receive data read from slave device
fOverlap	- TRUE for overlapped operation

This function is used to read a packet of data bytes from the specified TWI slave on the TWI bus associated with the port currently enabled on the specified interface handle. The TWI master will assert a START condition on the bus. If this succeeds, it will send the slave address plus read (SLA+R) and then attempt to read the specified number of bytes from the slave. After reading the bytes from the slave, a STOP condition will be asserted on the bus.

If the slave fails to ACK the address, the transaction will be aborted and an error returned. The TWI master will ACK each byte read from the slave, except the last byte, which will be NAK'd.

**DtwiMasterPutGet(HIF hif, BYTE dadr, DWORD cbSnd, BYTE \* rgbSnd, DWORD tusWait, DWORD cbRcv, BYTE \* rgbRcv, BOOL fOverlap)***Parameters*

hif	- open interface handle on the device
dadr	- device address of TWI slave device
cbSnd	- number of bytes to send to the slave device
rgbSnd	- buffer of data to send to the slave device
tusWait	- number of microseconds to wait after put before doing get
cbRcv	- number of bytes to read from the slave
rgbRcv	- buffer to receive data read from the slave
fOverlap	- TRUE for overlapped operation

This function is used to send a packet of data to a slave and read a resulting packet of data from the same slave as a single operation. It will write a packet of bytes to, and then read a packet of bytes from the specified TWI slave device on the TWI bus associated with the port currently enabled on the specified interface handle.

The TWI master will assert a START condition on the bus. If this succeeds, the slave address plus write (SLA+W) will be sent, followed by the bytes specified in *rgbSnd*.

After writing the data in *rgbSnd*, the device will wait for the amount of time specified by *tusWait* before beginning the read. The value in *tusWait* specifies the number of microseconds to wait. The maximum allowed wait value is 65535. A wait period of 0 is allowed.

Following the wait period, a repeated START will be asserted on the bus and the slave address plus read (SLA+R) will be sent. After this, the specified number of bytes will be read from the slave and returned in *rgbRcv*. After reading the bytes from the slave, a STOP condition will be asserted on the bus.

If the slave fails to ACK either the SLA+W or any of the data bytes sent, or if the slave fails to ACK the SLA+R, the transaction will be halted, a STOP condition asserted on the bus and an error code returned indicating what error occurred. During the read portion of the transaction, the TWI master will ACK each byte read from the slave, except the last byte, which will be NAK'd.

**DtwiMasterBatch**(HIF hif, DWORD cbSnd, BYTE \* rgbSnd, DWORD cbRcv, BYTE \* rgbRcv, BOOL fOverlap)*Parameters*

hif	- open interface handle on the device
cbSnd	- number of data bytes to send
rgbSnd	- pointer to data to send
cbRcv	- number of data bytes to be returned
rgbRcb	- buffer to receive returned data
fOverlap	- TRUE if operation should be overlapped

This function is used to initiate batch transactions on the TWI bus associated with the TWI port currently enabled on the specified interface handle. The *rgbSnd* buffer contains a batch command script specifying a series of transactions to occur on the bus. The device will interpret the contents of the buffer, performing the requested bus transactions.

The *cbSnd* and *rgbSnd* parameters specify the batch command buffer to be sent to the TWI master for execution. The *cbRcv* and *rgbRcv* parameters specify the expected number of bytes and the buffer to receive the bytes to be returned from the TWI master as a result of executing the commands in the batch script.

See the TWI Batch section of this document for a more complete description of TWI batch operations and the format of the batch command buffer.

## DTWI Slave API Functions

### **DtwiSlaveEnable(HIF hif, BYTE dadr, BOOL fGeneralCall, BOOL fAckAlways)**

#### *Parameters*

hif	- open interface handle on the device
dadr	- slave address for the device
fGeneralCall	- TRUE if device should respond to the general call address
fAckAlways	- TRUE if device should always ACK its address

This function is used to begin operation as a TWI slave device on the TWI bus associated with the currently enabled port on the specified interface handle. This function does nothing if the device does not support TWI slave capabilities.

If fGeneralCall is TRUE, the device will respond to the TWI general call (broadcast) address as well as its specified address. If fAckAlways is TRUE, the device will always ACK when its address is specified on the bus. If fAckAlways is FALSE, it will not ACK its address if its RX buffer is full for a write, or its TX buffer is empty for a read.

### **DtwiSlaveDisable(HIF hif)**

#### *Parameters*

hif	- open interface handle on the device
-----	---------------------------------------

This function is used to end operation as a TWI slave device on the TWI bus associated with the currently enabled port on the specified interface handle. Following this call, the TWI device associated with the port will no longer respond as a slave device on the bus.

Any unsent TX data or unread RX data in the buffers will be discarded.

### **DtwiSlaveQueryBuffer(HIF hif, DWORD \* pcbTx, DWORD \* pcbRx)**

#### *Parameters*

hif	- open interface handle on the device
pcbTx	- variable to receive size of TX buffer
pcbRx	- variable to receive size of RX buffer

This function will return the sizes of the transmit and receive buffers.

### **DtwiSlaveRxQuery(HIF hif, DWORD \* pcbRx)**

#### *Parameters*

hif	- open interface handle on the device
pcbRx	- variable to receive number of bytes in RX buffer

This function will return the number of bytes currently in the devices RX buffer. This is data written to the device by another master on the bus.

**DtwiSlaveRxRead(HIF hif, DWORD cbRcvMax, BYTE \* rgbRcv, DWORD \* pcbRcv, BOOL fOverlap)***Parameters*

hif	- open interface handle on the device
cbRcvMax	- maximum number of bytes to return
rgbRcv	- buffer to receive the bytes returned from the device
pcbRcv	- variable to receive actual number of bytes returned
fOverlap	- TRUE if operation should be overlapped

This function is used to read from the device slave receive buffer. The slave receive buffer will contain data written to the device by a master on the bus. If there is data waiting in the slave receive buffer, up to *cbBufMax* bytes will be returned in *rgbBuf*, and the actual number of byte returned will be placed in *\*pcbRcv*. If there is no data waiting, *\*pcbRcv* will be set to 0.

**DtwiSlaveTxQuery(HIF hif, DWORD \* pcbTx)***Parameters*

hif	- open interface handle on the device
pcbTx	- variable to receive number of bytes in slave TX buffer

This function will return the number of bytes currently in the slave TX buffer. The slave TX buffer holds data to be returned to master devices on the bus when they do a master read from the slave device.

**DtwiSlaveTxPost(HIF hif, DWORD cbTx, BYTE \* rgbTx, BOOL fOverlap)***Parameters*

hif	- open interface handle on the device
cbTx	- number of bytes to send to the device
rgbTx	- buffer containing data to send to the device
fOverlap	- TRUE if the operation should be overlapped

This function is used to post data to the slave transmit buffer. This data will then be available to transmit when some master on the bus attempts to read from the slave device.

## DTWI SMBus API Functions

### DtwiSmbQueryAlert(HIF hif, BOOL \* pfAlert)

#### Parameters

- hif - open interface handle on the device
- pfAlert - variable to receive alert state

This function will return the state of the SMBALERT# pin in the device. The return value will be TRUE if the pin is low (active state) or FALSE if the pin is high (inactive state).

### DtwiSmbSetSuspend(HIF hif, BOOL fSuspend)

#### Parameters

- hif - open interface handle on the device
- fSuspend - TRUE to activate suspend mode, FALSE to deactivate

This function will set the SMBSUS# pin on the device to the specified state. If fSuspend is TRUE, the pin will be set low (the active state). If fSuspend is FALSE, the pin will be set high (the inactive state).

### DtwiSmbPecEnable(HIF hif)

#### Parameters

- hif - open interface handle on the device

This function will enable SMBus packet error checking in the device.

### DtwiSmbPecDisable(HIF hif)

#### Parameters

- hif - open interface handle on the device

This function will disable SMBus packet error checking in the device.