

Multi parameter IO Module

ADC

This USB powered module is designed to interface the Northern Scientific models NS 621 and NS623 (1024 and 8192 channel ADCs) and the Tracor Northern TN 12xx series of ADCs (1024 - 8192 channels).

The IO Module may be daisy chained to include up to 8 IO modules in total. The modules may be configured for independent or multiparameter acquisition.

In independent mode each module transfers data through its independent USB interface.

In multiparameter mode the data transfers are synchronised and use the USB interface of the first multiparameter configured interface in the IO Module Chain. Both independent and multiparameter mode acquisitions may be simultaneously active in a chain of IO Modules.

22 May 2011
ART

The data transfer rate is up to 1 MB per second - 500K ADC conversions per second. Data is transferred from the ADC to the USB as two bytes per conversion with the LSB first followed by the MSB. The Data format is:

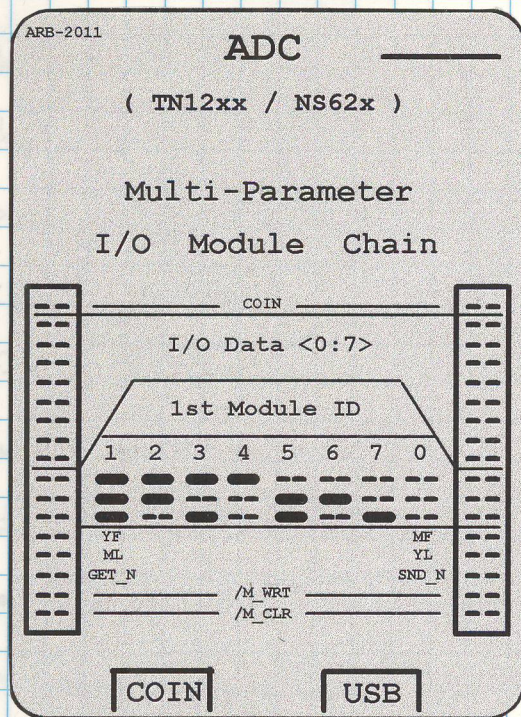
ADC Bits <7:0>

LSB	7	6	5	4	3	2	1	0
-----	---	---	---	---	---	---	---	---

MSB	ID <2:0>			ADC Bits <12:8>				
	ID2	ID1	ID0	12	11	10	9	8

ID <2:0> contain the three Bit code identifying the IO Module.

The ID of the first IO Module is by default 0. However the ID can be changed by using jumpers on the first module's daisy chain cable as indicated by the legend on the IO Module. Module IDs increase from Right to Left along the daisy chain of IO Modules.



The BNC COIN Connector requires a 5 volt pulse (or TTL level) to activate the multiparameter Acquisition Mode. Only one (1) BNC COIN input is required to activate all multiparameter enabled IO Modules in the Chain. The COIN signal is routed through the daisy chain cable.

Parts List

1	Plastic Case	Hammond	RL6215	80x120 mm
1	Isolated BNC Jack			
1	84 pin PLCC Through hole Socket			Mill MAX
2	2x17 pin Shrouded Header	Sullins	SBH-PBPC-D17-ST-BK	
1	2x10 pin Shrouded Header	Sullins	SBH-PBPC-D10-ST-BK	
3	SN74LVC2G13Z DCT	Dual Input Schmitt trigger	NAND Gates	
1	XC9572	Xilinx	Programmable CPLD	(-ISC)
1	DLP-USB245M	USB to FIFO Parallel Interface	Module	
1	IDC Cable	HSC34H/AE34G/HSC34H	6"	Recep - Recep 34pin
1	IDC Cable	HSC34H/AE34G/HPH34H	6"	Recep - Plug 34pin
1	IDC Cable	HSC20H/AE20G/HPH20H	6"	Recep - Plug 20pin
3	10 pF Cap Ceramic	0805	SMD	
4	.1mF Cap Ceramic	0805	SMD	
5	.1mF Cap Ceramic	1206	SMD	
1	220 pF Cap Ceramic	0805	SMD	
1	270 pF Cap Ceramic	0805	SMD	
1	4700 pF Cap Ceramic	0805	SMD	
11	120 ohm 1/8 watt	0805	SMD	
3	330 ohm 1/8 watt	0805	SMD	
10	470 ohm 1/8 watt	0805	SMD	
1	510 ohm 1/8 watt	0805	SMD	

7	4.7K ohm	1/8 watt	0805	SMD
15	10K ohm	1/8 watt	0805	SMD
2	47K ohm	1/8 watt	0805	SMD

TN12xx ADC Cable Parts

1	37 pin D-Sub Connector
1	Cinch Deep Straight Shield and Clamp Assembly (Metal)
1	37 pos Slide Latch Kit (Amplimite - Tyco)
1	20pin IDC Cable, plug on one end

External Coincidence

Multiple ADC Data Transfer Interface

ADC Interface

USB Interface

Xilinx 9572 84 Pin PLCC

Unused Pins:

7, 36, 37, 39, 40
41, 50, 75, 76

Device Programming Header

Read Clock

Timeout Clock

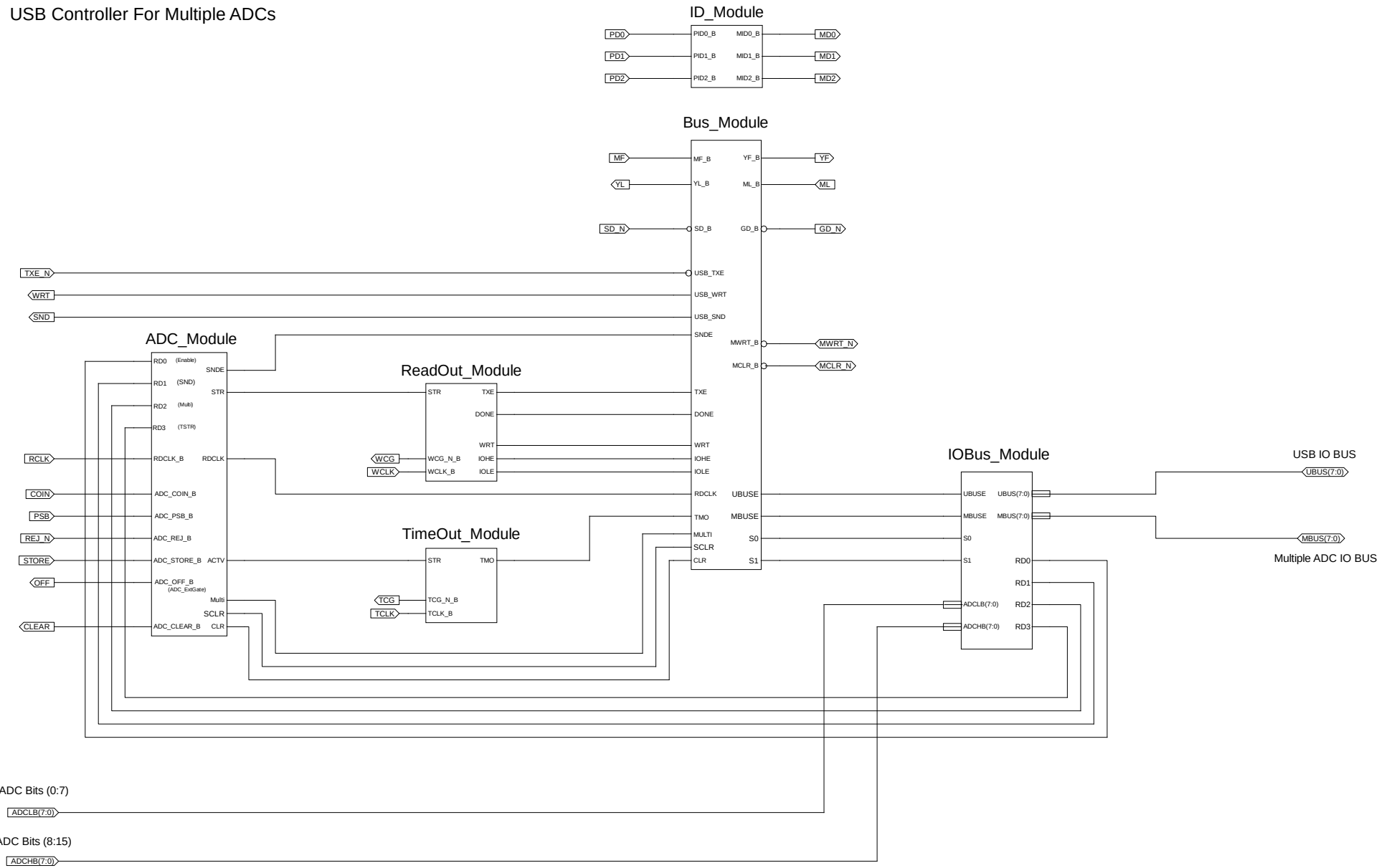
Write Clock

Kent State University

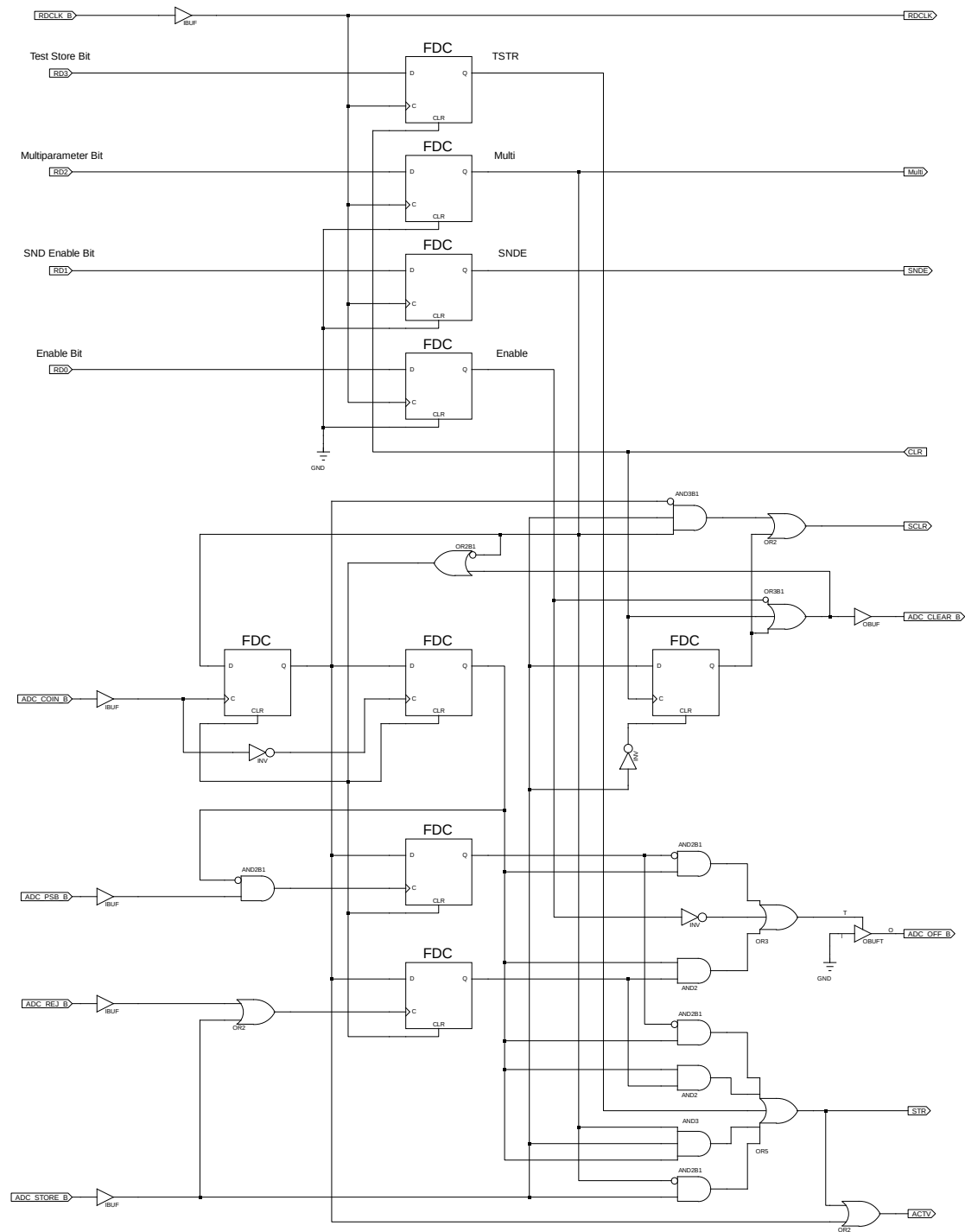
USB - ADC Controller

Alan R. Baldwin Rev. 1.03 4/4/2011 Page 1 of 1

USB Controller For Multiple ADCs

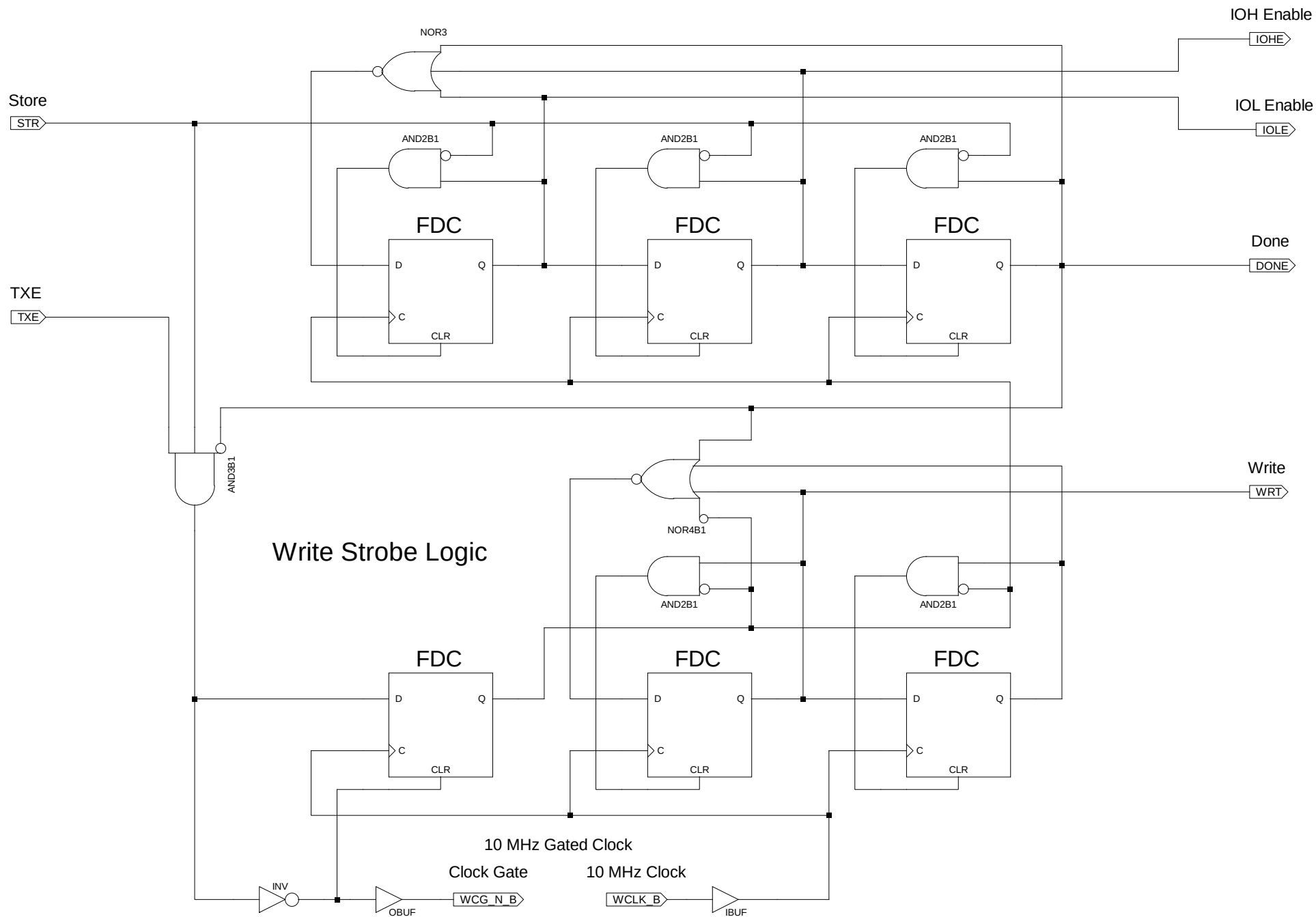


ADC Module

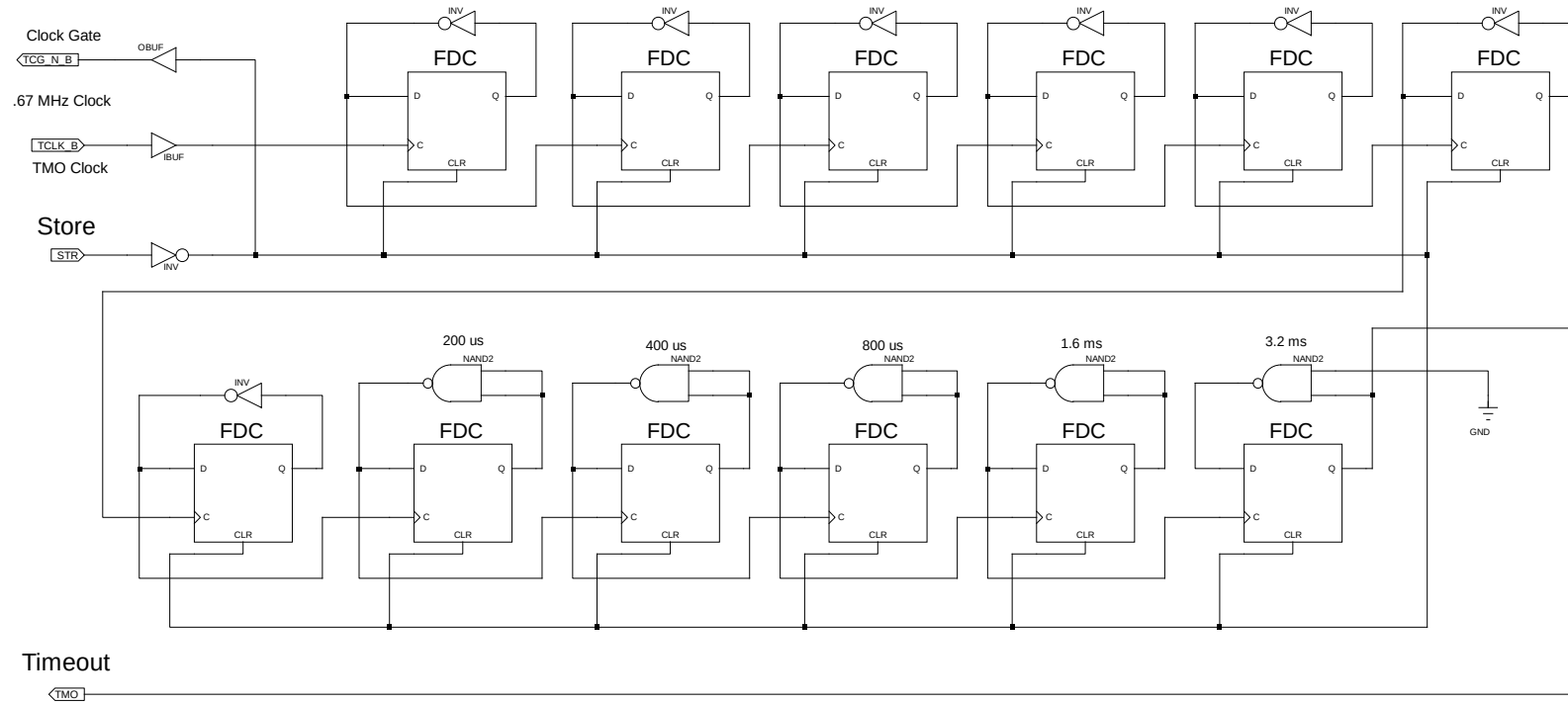


ReadOut Module

Readout Sequencer

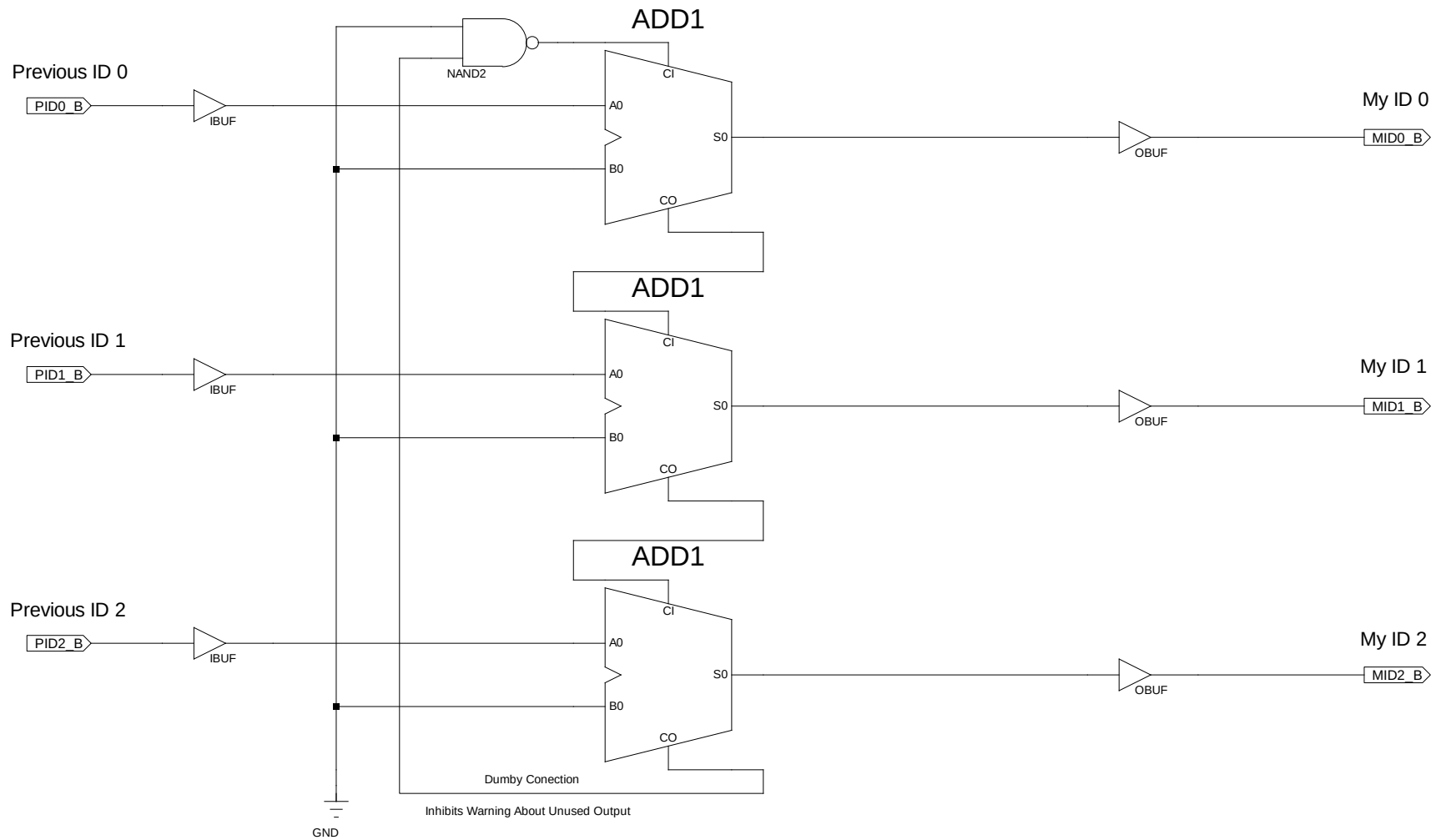


Timeout Module

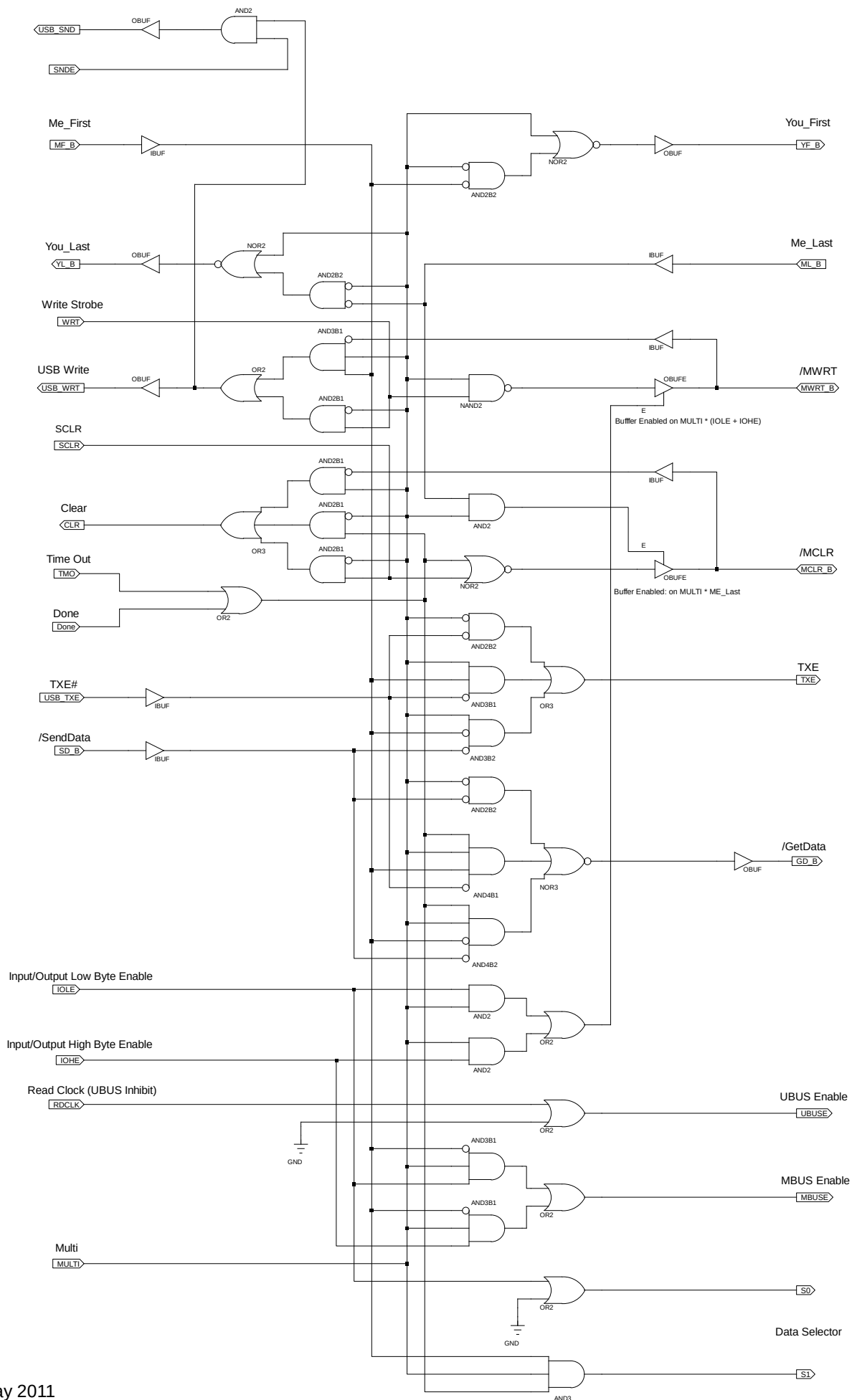
USB ADC Control Logic Element
September 2011

Symbol: TIMEOUT_MODULE

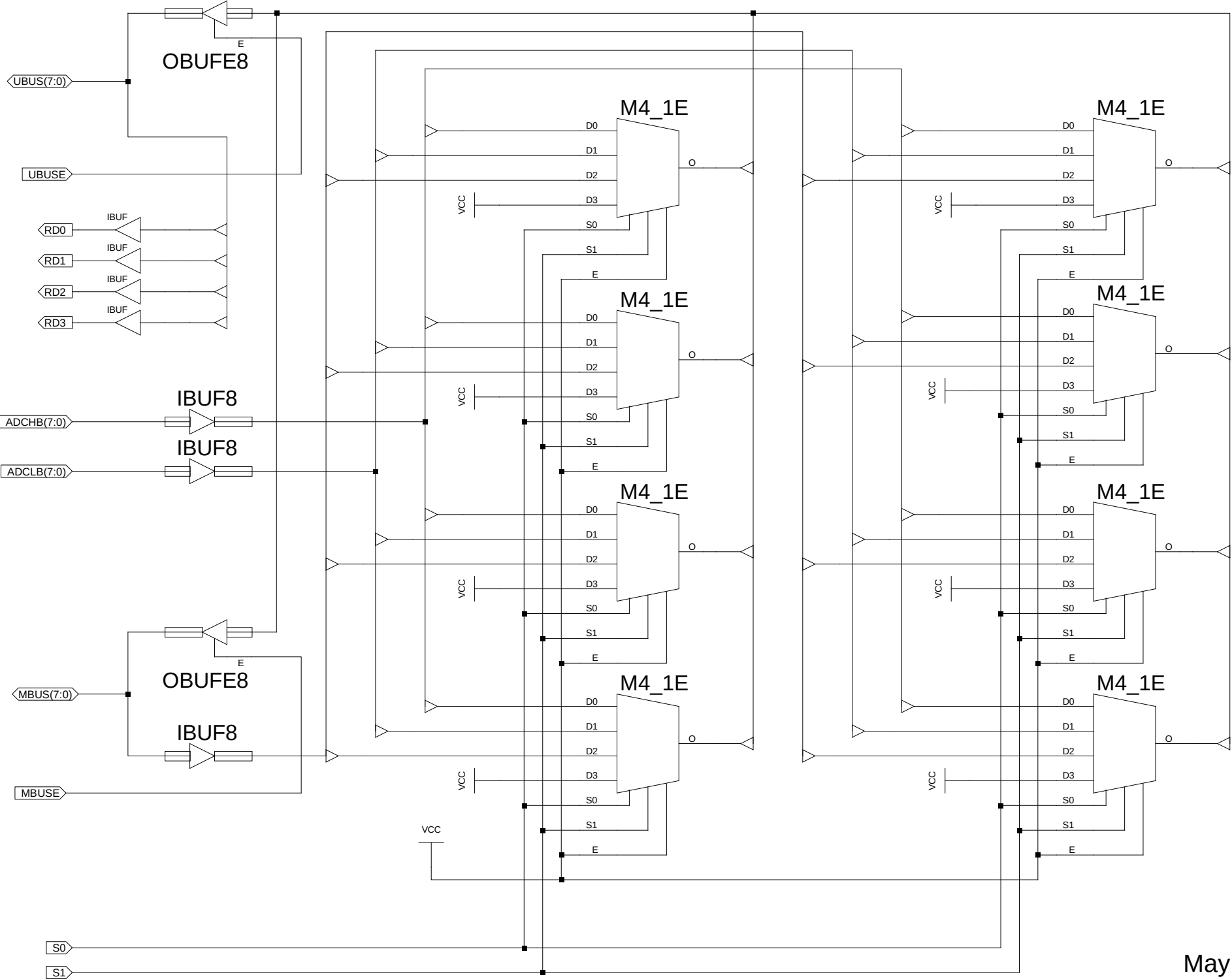
Interface ID Module



Multiple ADC Bus Module Control



IO Bus Module



Summary

Design Name	USB_ADC_Controller_Multiple
Fitting Status	Successful
Software Version	O.40d
Device Used	XC9572-15-PC84
Date	9-28-2011, 3:29PM

RESOURCES SUMMARY

Macrocells Used	Pterms Used	Registers Used	Pins Used	Function Block Inputs Used
64/72 (89%)	148/360 (42%)	27/72 (38%)	60/69 (87%)	121/144 (85%)

PIN RESOURCES

Signal Type	Required	Mapped	Pin Type	Used	Total
Input	27	27	I/O	55	64
Output	16	16	GCK/IO	3	3
Bidirectional	14	14	GTS/IO	1	2
GCK	3	3	GSR/IO	1	1
GTS	0	0			
GSR	0	0			

GLOBAL RESOURCES

Signal mapped onto global clock net (GCK1)	WCLK
Signal mapped onto global clock net (GCK2)	TCLK
Signal mapped onto global clock net (GCK3)	RCLK

POWER DATA

Macrocells in high performance mode (MCHP)	64
Macrocells in low power mode (MCLP)	0
Total macrocells used (MC)	64

[back to top](#)
[print page](#)

cpldfit: version 0.40d

Xilinx Inc.

Fitter Report

Design Name: USB_ADC_Controller_Multiple

Date: 9-28-2011, 3:29PM

Device Used: XC9572-15-PC84

Fitting Status: Successful

***** Mapped Resource Summary *****

Macrocells	Product Terms	Function Block	Registers	Pins
Used/Tot	Used/Tot	Inps Used/Tot	Used/Tot	Used/Tot
64 /72 (89%)	148 /360 (41%)	121/144 (84%)	27 /72 (37%)	60 /69 (87%)

** Function Block Resources **

Function Block	Mcells Used/Tot	FB Inps Used/Tot	Signals Used	Pterms Used/Tot	IO Used/Tot
FB1	16/18	31/36	31	42/90	9/18
FB2	16/18	32/36	32	39/90	11/17
FB3	16/18	31/36	31	37/90	8/17
FB4	16/18	27/36	27	30/90	2/17
	-----	-----		-----	-----
	64/72	121/144		148/360	30/69

* - Resource is exhausted

** Global Control Resources **

Signal 'WCLK' mapped onto global clock net GCK1.

Signal 'TCLK' mapped onto global clock net GCK2.

Signal 'RCLK' mapped onto global clock net GCK3.

Global output enable net(s) unused.

Global set/reset net(s) unused.

** Pin Resources **

Signal Type	Required	Mapped	Pin Type	Used	Total
Input	: 27	27	I/O	: 55	63
Output	: 16	16	GCK/IO	: 3	3
Bidirectional	: 14	14	GTS/IO	: 1	2
GCK	: 3	3	GSR/IO	: 1	1
GTS	: 0	0			
GSR	: 0	0			
	----	----			
Total	60	60			

** Power Data **

There are 64 macrocells in high performance mode (MCHP).

There are 0 macrocells in low power mode (MCLP).

End of Mapped Resource Summary

□***** Errors and Warnings *****

WARNING:Cpld - Unable to retrieve the path to the iSE Project Repository. Will use the default filename of 'USB_ADC_Controller_Multiple.ise'.

INFO:Cpld - Inferring BUFG constraint for signal 'XLXI_3/WCLK' based upon the LOC constraint 'P9'. It is recommended that you declare this BUFG explicitly in your design. Note that for certain device families the output of a BUFG constraint can not drive a gated clock, and the BUFG constraint

will be ignored.

INFO:Cpld - Inferring BUFG constraint for signal 'XLXI_4/TCLK' based upon the LOC constraint 'P10'. It is recommended that you declare this BUFG explicitedly in your design. Note that for certain device families the output of a BUFG constraint can not drive a gated clock, and the BUFG constraint will be ignored.

INFO:Cpld - Inferring BUFG constraint for signal 'XLXI_15/RCLK' based upon the LOC constraint 'P12'. It is recommended that you declare this BUFG explicitedly in your design. Note that for certain device families the output of a BUFG constraint can not drive a gated clock, and the BUFG constraint will be ignored.

□***** Summary of Mapped Logic *****

** 30 Outputs **

Signal Name	Total Pts	Total Inps	Loc	Pin No.	Pin Type	Pin Use	Pwr Mod
MWRT_N	2	3	FB1_1	4	I/O	I/O	STD
YF	1	2	FB1_2	1	I/O	O	STD
GD_N	4	6	FB1_6	3	I/O	O	STD
WCG	3	6	FB1_7	11	I/O	O	STD
MCLR_N	4	7	FB1_8	5	I/O	I/O	STD
TCG	1	2	FB1_10	13	I/O	O	STD
YL	1	2	FB1_15	14	I/O	O	STD
UBUS<7>	4	6	FB1_16	23	I/O	O	STD
UBUS<6>	4	6	FB1_18	24	I/O	O	STD
CLEAR	1	3	FB2_1	63	I/O	O	STD
MBUS<4>	2	4	FB2_2	69	I/O	I/O	STD
MBUS<6>	2	4	FB2_3	67	I/O	I/O	STD
MBUS<5>	4	8	FB2_4	68	I/O	I/O	STD
MBUS<3>	4	8	FB2_5	70	I/O	I/O	STD
MBUS<2>	2	4	FB2_6	71	I/O	I/O	STD
MBUS<1>	4	8	FB2_8	72	I/O	I/O	STD
MBUS<0>	4	8	FB2_9	74	GSR/I/O	I/O	STD
MD1	2	2	FB2_15	83	I/O	O	STD
MD0	1	1	FB2_16	82	I/O	O	STD
MD2	2	3	FB2_17	84	I/O	O	STD
UBUS<5>	2	2	FB3_1	25	I/O	O	STD
UBUS<3>	2	2	FB3_3	31	I/O	I/O	STD
UBUS<2>	4	6	FB3_4	32	I/O	I/O	STD
UBUS<0>	4	6	FB3_6	34	I/O	I/O	STD
SND	2	5	FB3_7	35	I/O	O	STD
WRT	2	4	FB3_8	21	I/O	O	STD
UBUS<4>	4	6	FB3_9	26	I/O	O	STD
UBUS<1>	4	6	FB3_11	33	I/O	I/O	STD
OFF	1	2	FB4_12	58	I/O	O	STD
MBUS<7>	2	4	FB4_17	66	I/O	I/O	STD

** 34 Buried Nodes **

Signal Name	Total Pts	Total Inps	Loc	Pwr Mode	Reg State	Init
XLXN_88	2	4	FB1_5	STD	RESET	
XLXI_3/XLXN_506	2	3	FB1_9	STD	RESET	
\$OpTx\$FX_SC\$9	2	5	FB1_11	STD		
XLXN_92	3	4	FB1_12	STD	RESET	
XLXI_15/XLXN_49	3	4	FB1_13	STD	RESET	
XLXI_15/XLXN_286	3	3	FB1_14	STD	RESET	
XLXI_15/XLXN_27	3	3	FB1_17	STD	RESET	
XLXI_15/XLXN_42/XLXI_15/XLXN_42_RSTF	1	4	FB2_11	STD		

XLXI_15/XLXN_213/XLXI_15/XLXN_213_RSTF	2	5	FB2_12	STD	
\$OpTx\$FX_DC\$3	2	4	FB2_13	STD	
XLXN_131	3	5	FB2_14	STD	RESET
XLXN_130	3	4	FB2_18	STD	RESET
XLXN_149	1	1	FB3_10	STD	RESET
XLXN_123	1	1	FB3_12	STD	RESET
XLXI_4/XLXN_79	1	2	FB3_13	STD	RESET
XLXN_96	2	3	FB3_14	STD	RESET
XLXI_4/XLXN_85	2	3	FB3_15	STD	RESET
XLXI_4/XLXN_78	2	3	FB3_16	STD	RESET
XLXI_4/XLXN_35	2	3	FB3_17	STD	RESET
XLXI_4/XLXN_299	2	3	FB3_18	STD	RESET
XLXI_15/XLXN_42/XLXI_15/XLXN_42_CLKF	1	2	FB4_3	STD	
XLXI_15/XLXN_239	1	1	FB4_4	STD	RESET
\$OpTx\$FX_DC\$11	1	2	FB4_5	STD	
XLXI_4/XLXN_298	2	3	FB4_6	STD	RESET
XLXI_4/XLXN_296	2	3	FB4_7	STD	RESET
XLXI_4/XLXN_295	2	3	FB4_8	STD	RESET
XLXI_4/XLXN_277	2	3	FB4_9	STD	RESET
XLXI_4/XLXN_275	2	3	FB4_10	STD	RESET
XLXI_4/XLXN_143	2	3	FB4_11	STD	RESET
XLXI_3/XLXN_595	2	1	FB4_13	STD	RESET
XLXI_15/XLXN_213	2	2	FB4_14	STD	RESET
\$OpTx\$FX_DC\$4	2	3	FB4_15	STD	
XLXI_15/XLXN_42	3	3	FB4_16	STD	RESET
XLXI_15/XLXN_248	3	2	FB4_18	STD	RESET

** 30 Inputs **

Signal	Loc	Pin	Pin	Pin
Name		No.	Type	Use
SD_N	FB1_3	6	I/O	I
ML	FB1_5	2	I/O	I
WCLK	FB1_9	9	GCK/I/O	GCK
TCLK	FB1_11	10	GCK/I/O	GCK
PD1	FB1_12	18	I/O	I
TXE_N	FB1_13	20	I/O	I
RCLK	FB1_14	12	GCK/I/O	GCK/I
MF	FB1_17	15	I/O	I
COIN	FB2_11	77	GTS/I/O	I
ADCHB<5>	FB2_12	79	I/O	I
ADCHB<6>	FB2_13	80	I/O	I
ADCHB<7>	FB2_14	81	I/O	I
PD2	FB3_2	17	I/O	I
PD0	FB3_5	19	I/O	I
ADCLB<0>	FB3_13	43	I/O	I
ADCLB<2>	FB3_16	45	I/O	I
ADCLB<3>	FB4_1	46	I/O	I
ADCLB<1>	FB4_2	44	I/O	I
ADCLB<6>	FB4_3	51	I/O	I
ADCLB<7>	FB4_4	52	I/O	I
ADCLB<4>	FB4_5	47	I/O	I
ADCHB<1>	FB4_6	54	I/O	I
ADCHB<2>	FB4_7	55	I/O	I
ADCLB<5>	FB4_8	48	I/O	I
ADCHB<4>	FB4_10	57	I/O	I
ADCHB<0>	FB4_11	53	I/O	I
STORE	FB4_13	61	I/O	I
ADCHB<3>	FB4_14	56	I/O	I
REJ_N	FB4_15	65	I/O	I

PSB

FB4_16 62 I/O I

Legend:

Pin No. - ~ - User Assigned

□***** Function Block Details *****

Legend:

Total Pt - Total product terms used by the macrocell signal

Imp Pt - Product terms imported from other macrocells

Exp Pt - Product terms exported to other macrocells
in direction shown

Unused Pt - Unused local product terms remaining in macrocell

Loc - Location where logic was mapped in device

Pin Type/Use - I - Input GCK - Global Clock
O - Output GTS - Global Output Enable
(b) - Buried macrocell GSR - Global Set/ResetX(@) - Signal used as input (wire-AND input) to the macrocell logic.
The number of Signals Used may exceed the number of FB Inputs
Used due to wire-ANDing in the switch matrix.

Pin No. - ~ - User Assigned

□***** FB1 *****

Number of function block inputs used/remaining: 31/5

Number of signals used by logic mapping into function block: 31

Signal Name	Total Pt	Imp Pt	Exp Pt	Unused Pt	Loc	Pin #	Pin Type	Pin Use
MWRT_N	2	0	0	3	FB1_1	4	I/O	I/O
YF	1	0	0	4	FB1_2	1	I/O	O
(unused)	0	0	0	5	FB1_3	6	I/O	I
(unused)	0	0	0	5	FB1_4	7	I/O	
XLXN_88	2	0	0	3	FB1_5	2	I/O	I
GD_N	4	0	0	1	FB1_6	3	I/O	O
WCG	3	0	0	2	FB1_7	11	I/O	O
MCLR_N	4	0	0	1	FB1_8	5	I/O	I/O
XLXI_3/XLXN_506	2	0	0	3	FB1_9	9	GCK/I/O	GCK
TCG	1	0	0	4	FB1_10	13	I/O	O
\$OpTx\$FX_SC\$9	2	0	0	3	FB1_11	10	GCK/I/O	GCK
XLXN_92	3	0	0	2	FB1_12	18	I/O	I
XLXI_15/XLXN_49	3	0	0	2	FB1_13	20	I/O	I
XLXI_15/XLXN_286	3	0	0	2	FB1_14	12	GCK/I/O	GCK/I
YL	1	0	0	4	FB1_15	14	I/O	O
UBUS<7>	4	0	0	1	FB1_16	23	I/O	O
XLXI_15/XLXN_27	3	0	0	2	FB1_17	15	I/O	I
UBUS<6>	4	0	0	1	FB1_18	24	I/O	O

Signals Used by Logic in Function Block

1: \$OpTx\$FX_DC\$11	12: XLXI_15/XLXN_42/XLXI_15/XLXN_42_RSTF	22: TXE_N
2: \$OpTx\$FX_DC\$3	13: ADCHB<6>	23: MF
3: \$OpTx\$FX_DC\$4	14: ADCHB<7>	24: SD_N
4: \$OpTx\$FX_SC\$9.LFBK	15: ADCLB<6>	25: XLXN_123
5: XLXI_15/XLXN_213	16: ADCLB<7>	26: XLXN_130
6: XLXI_15/XLXN_248	17: MBUS<6>.PIN	27: XLXN_131
7: STORE	18: MBUS<7>.PIN	28: RCLK
8: XLXI_15/XLXN_27.LFBK	19: XLXI_3/XLXN_506.LFBK	29: XLXN_88.LF
9: PSB	20: XLXI_3/XLXN_595	30: XLXN_92.LF
10: XLXI_15/XLXN_286.LFBK	21: ML	31: XLXN_96
11: COIN		

Signal Name	1	2	3	4	Signals FB	Used	Inputs
MWRT_N	X.....	X...X.....			3		3
YF		X.X.....			2		2

```

XLXN_88      .....XX.....XX..... 4      4
GD_N         .....XXXX....XX..... 6      6
WCG          ...X.....XXXX....X..... 6      6
MCLR_N       .....XXX.....X...X...XX..... 7      7
XLXI_3/XLXN_506 .....XX.....X..... 3      3
TCG          ...X...X..... 2      2
$OpTx$FX_SC$9 ..X.X.X.X.....X..... 5      5
XLXN_92      ...X.....X....X...X..... 4      4
XLXI_15/XLXN_49 .....XXX.X..... 4      4
XLXI_15/XLXN_286 .....X..XX..... 3      3
YL           .....X...X..... 2      2
UBUS<7>      .X.....X.X.X.....XX..... 6      6
XLXI_15/XLXN_27 .....XX.....X..... 3      3
UBUS<6>      .X.....X.X.X.....XX..... 6      6
0-----+-----1-----+-----2-----+-----3-----+-----4
0          0          0          0

```

□***** FB2 *****

Number of function block inputs used/remaining: 32/4

Number of signals used by logic mapping into function block: 32

Signal Name	Total Pt	Imp Pt	Exp Pt	Unused Pt	Loc	Pin #	Pin Type	Pin Use
CLEAR	1	0	0	4	FB2_1	63	I/O	0
MBUS<4>	2	0	0	3	FB2_2	69	I/O	I/O
MBUS<6>	2	0	0	3	FB2_3	67	I/O	I/O
MBUS<5>	4	0	0	1	FB2_4	68	I/O	I/O
MBUS<3>	4	0	0	1	FB2_5	70	I/O	I/O
MBUS<2>	2	0	0	3	FB2_6	71	I/O	I/O
(unused)	0	0	0	5	FB2_7	76	GTS/I/O	
MBUS<1>	4	0	0	1	FB2_8	72	I/O	I/O
MBUS<0>	4	0	0	1	FB2_9	74	GSR/I/O	I/O
(unused)	0	0	0	5	FB2_10	75	I/O	
XLXI_15/XLXN_42/XLXI_15/XLXN_42_RSTF	1	0	0	4	FB2_11	77	GTS/I/O	I
XLXI_15/XLXN_213/XLXI_15/XLXN_213_RSTF	2	0	0	3	FB2_12	79	I/O	I
\$OpTx\$FX_DC\$3	2	0	0	3	FB2_13	80	I/O	I
XLXN_131	3	0	0	2	FB2_14	81	I/O	I
MD1	2	0	0	3	FB2_15	83	I/O	0
MD0	1	0	0	4	FB2_16	82	I/O	0
MD2	2	0	0	3	FB2_17	84	I/O	0
XLXN_130	3	0	0	2	FB2_18		(b)	(b)

Signals Used by Logic in Function Block

1: \$OpTx\$FX_DC\$11	12: PD2	23: MBUS<3>.
2: \$OpTx\$FX_DC\$3.LFBK	13: ADCHB<0>	24: MBUS<5>.
3: \$OpTx\$FX_SC\$9	14: ADCHB<1>	25: XLXI_3/X
4: UBUS<2>	15: ADCHB<3>	26: MCLR_N.P
5: UBUS<4>	16: ADCHB<5>	27: MF
6: UBUS<6>	17: ADCLB<0>	28: XLXN_123
7: XLXI_15/XLXN_213/XLXI_15/XLXN_213_RSTF.LFBK	18: ADCLB<1>	29: XLXN_130
8: XLXI_15/XLXN_239	19: ADCLB<3>	30: XLXN_131
9: XLXI_15/XLXN_248	20: ADCLB<5>	31: XLXN_92
10: PD0	21: MBUS<0>.PIN	32: XLXN_96
11: PD1	22: MBUS<1>.PIN	

Signal Name	1	2	3	4	Signals	FB
CLEAR	0-----+-----0-----+-----0-----+-----0-----+-----0				Used	Inputs
MBUS<4>	XXX.....				3	3
MBUS<6>	X...X.....XX.....				4	4
MBUS<6>	X....X.....XX.....				4	4


```

MBUS<5>          XX.....X...X...X..XX.X..... 8      8
MBUS<3>          XX.....X...X...X...XX.X..... 8      8
MBUS<2>          X..X.....XX..... 4      4
MBUS<1>          XX.....X...X...X...XX.X..... 8      8
MBUS<0>          XX.....X...X...X...XX.X..... 8      8
XLXI_15/XLXN_42/XLXI_15/XLXN_42_RSTF
.....XX.....X.X..... 4      4
XLXI_15/XLXN_213/XLXI_15/XLXN_213_RSTF
.....X.....X.X..XX..... 5      5
$OpTx$FX_DC$3   .....XX..XX..... 4      4
XLXN_131        X.X.....X...XX..... 5      5
MD1             .....XX..... 2      2
MD0             .....X..... 1      1
MD2             .....XXX..... 3      3
XLXN_130        ..X.....X...XX..... 4      4
0-----+-----1-----+-----2-----+-----3-----+-----4
0          0          0          0          0
□***** FB3 *****

```

Number of function block inputs used/remaining: 31/5

Number of signals used by logic mapping into function block: 31

Signal Name	Total Pt	Imp Pt	Exp Pt	Unused Pt	Loc	Pin #	Pin Type	Pin Use
UBUS<5>	2	0	0	3	FB3_1	25	I/O	O
(unused)	0	0	0	5	FB3_2	17	I/O	I
UBUS<3>	2	0	0	3	FB3_3	31	I/O	I/O
UBUS<2>	4	0	0	1	FB3_4	32	I/O	I/O
(unused)	0	0	0	5	FB3_5	19	I/O	I
UBUS<0>	4	0	0	1	FB3_6	34	I/O	I/O
SND	2	0	0	3	FB3_7	35	I/O	O
WRT	2	0	0	3	FB3_8	21	I/O	O
UBUS<4>	4	0	0	1	FB3_9	26	I/O	O
XLXN_149	1	0	0	4	FB3_10	40	I/O	(b)
UBUS<1>	4	0	0	1	FB3_11	33	I/O	I/O
XLXN_123	1	0	0	4	FB3_12	41	I/O	(b)
XLXI_4/XLXN_79	1	0	0	4	FB3_13	43	I/O	I
XLXN_96	2	0	0	3	FB3_14	36	I/O	(b)
XLXI_4/XLXN_85	2	0	0	3	FB3_15	37	I/O	(b)
XLXI_4/XLXN_78	2	0	0	3	FB3_16	45	I/O	I
XLXI_4/XLXN_35	2	0	0	3	FB3_17	39	I/O	(b)
XLXI_4/XLXN_299	2	0	0	3	FB3_18		(b)	(b)

Signals Used by Logic in Function Block

1: \$OpTx\$FX_DC\$3	12: ADCLB<2>	22: XLXI_4/XLXN_79.LFBK
2: \$OpTx\$FX_SC\$9	13: ADCLB<4>	23: MWRT_N.PIN
3: MBUS<3>	14: MBUS<0>.PIN	24: MF
4: MBUS<5>	15: MBUS<1>.PIN	25: XLXN_123.LFBK
5: XLXI_15/XLXN_27	16: MBUS<2>.PIN	26: XLXN_131
6: ADCHB<0>	17: MBUS<4>.PIN	27: UBUS<2>.PIN
7: ADCHB<1>	18: XLXI_4/XLXN_277	28: UBUS<1>.PIN
8: ADCHB<2>	19: XLXI_4/XLXN_296	29: XLXN_149.LFBK
9: ADCHB<4>	20: XLXI_4/XLXN_298	30: RCLK
10: ADCLB<0>	21: XLXI_4/XLXN_78.LFBK	31: XLXN_88
11: ADCLB<1>		

Signal Name	1	2	3	4	Signals	FB
UBUS<5>	0-----+-----	0-----+-----	0-----+-----	0-----+-----	Used	Inputs
UBUS<3>	...X.....		...X.....		2	2
UBUS<2>	..X.....		..X.....		2	2
UBUS<1>	X.....X...X...X.....		X...X.....		6	6
UBUS<0>	X....X...X...X.....		X...X.....		6	6

```

SND                .....XXX...X.X..... 5      5
WRT                .....XXX...X..... 4      4
UBUS<4>            X.....X...X...X.....X...X..... 6      6
XLXN_149           .....X..... 1      1
UBUS<1>            X.....X...X...X.....X...X..... 6      6
XLXN_123           .....X..... 1      1
XLXI_4/XLXN_79     .X..X..... 2      2
XLXN_96            .X..X.....X..... 3      3
XLXI_4/XLXN_85     .X..X.....X..... 3      3
XLXI_4/XLXN_78     .X..X.....X..... 3      3
XLXI_4/XLXN_35     .X..X.....X..... 3      3
XLXI_4/XLXN_299    .X..X.....X..... 3      3

```

```

0-----+-----1-----+-----2-----+-----3-----+-----4
0              0              0              0

```

□***** FB4 *****

Number of function block inputs used/remaining: 27/9

Number of signals used by logic mapping into function block: 27

Signal Name	Total Pt	Imp Pt	Exp Pt	Unused Pt	Loc	Pin #	Pin Type	Pin Use
(unused)	0	0	0	5	FB4_1	46	I/O	I
(unused)	0	0	0	5	FB4_2	44	I/O	I
XLXI_15/XLXN_42/XLXI_15/XLXN_42_CLKF	1	0	0	4	FB4_3	51	I/O	I
XLXI_15/XLXN_239	1	0	0	4	FB4_4	52	I/O	I
\$OpTx\$FX_DC\$11	1	0	0	4	FB4_5	47	I/O	I
XLXI_4/XLXN_298	2	0	0	3	FB4_6	54	I/O	I
XLXI_4/XLXN_296	2	0	0	3	FB4_7	55	I/O	I
XLXI_4/XLXN_295	2	0	0	3	FB4_8	48	I/O	I
XLXI_4/XLXN_277	2	0	0	3	FB4_9	50	I/O	(b)
XLXI_4/XLXN_275	2	0	0	3	FB4_10	57	I/O	I
XLXI_4/XLXN_143	2	0	0	3	FB4_11	53	I/O	I
OFF	1	0	0	4	FB4_12	58	I/O	O
XLXI_3/XLXN_595	2	0	0	3	FB4_13	61	I/O	I
XLXI_15/XLXN_213	2	0	0	3	FB4_14	56	I/O	I
\$OpTx\$FX_DC\$4	2	0	0	3	FB4_15	65	I/O	I
XLXI_15/XLXN_42	3	0	0	2	FB4_16	62	I/O	I
MBUS<7>	2	0	0	3	FB4_17	66	I/O	I/O
XLXI_15/XLXN_248	3	0	0	2	FB4_18		(b)	(b)

Signals Used by Logic in Function Block

1: \$OpTx\$FX_DC\$11.LFBK	10: REJ_N
2: \$OpTx\$FX_DC\$4.LFBK	11: XLXI_15/XLXN_286
3: \$OpTx\$FX_SC\$9	12: XLXI_15/XLXN_42.LFBK
4: UBUS<7>	13: XLXI_15/XLXN_42/XLXI_15/XLXN_42_CLK
5: WCG	14: XLXI_15/XLXN_42/XLXI_15/XLXN_42_RST
6: XLXI_15/XLXN_213/XLXI_15/XLXN_213_RSTF	15: XLXI_15/XLXN_49
7: XLXI_15/XLXN_239.LFBK	16: XLXI_4/XLXN_143.LFBK
8: STORE	17: XLXI_4/XLXN_275.LFBK
9: XLXI_15/XLXN_27	18: XLXI_4/XLXN_295.LFBK

Signal Name	1	2	3	4	Signals Used	FB Inputs
XLXI_15/XLXN_42/XLXI_15/XLXN_42_CLKF	0-----+-----0-----+-----0-----+-----0					
	X.X.....				2	2
XLXI_15/XLXN_239	X.....				1	1
\$OpTx\$FX_DC\$11	XX.....				2	2
XLXI_4/XLXN_298	..X.....X.....X.....				3	3
XLXI_4/XLXN_296	..X.....X.....X.....				3	3
XLXI_4/XLXN_295	..X.....X.....X.....				3	3
XLXI_4/XLXN_277	..X.....X.....X.....				3	3

XLXI_4/XLXN_275	..X.....X.....X.....	3	3
XLXI_4/XLXN_143	..X.....X.....X.....	3	3
OFF	.X....X.....	2	2
XLXI_3/XLXN_595	X.....	1	1
XLXI_15/XLXN_213	X.....X.....	2	2
\$OpTx\$FX_DC\$4	XX..X.....	3	3
XLXI_15/XLXN_42	X..XX.....	3	3
MBUS<7>	X..X.....XX.....	4	4
XLXI_15/XLXN_248	X.X.....	2	2
	0----+----1----+----2----+----3----+----4		
	0 0 0 0		

□***** Equations *****

***** Mapped Logic *****

```
$OpTx$FX_DC$11 <= (NOT XLXN_131 AND NOT XLXN_130);
```

```
$OpTx$FX_DC$3 <= ((MF AND XLXN_92 AND XLXN_123)
OR (MF AND XLXN_123 AND XLXN_96));
```

```
$OpTx$FX_DC$4 <= ((NOT XLXI_15/XLXN_286)
OR (XLXI_15/XLXN_49 AND NOT XLXI_15/XLXN_42.LFBK));
```

```
$OpTx$FX_SC$9 <= ((NOT XLXI_15/XLXN_213 AND NOT STORE AND $OpTx$FX_DC$4)
OR (NOT XLXI_15/XLXN_213 AND XLXN_123 AND
NOT XLXI_15/XLXN_286.LFBK));
```

```
CLEAR <= NOT ((NOT XLXI_15/XLXN_248 AND XLXI_15/XLXN_239 AND
XLXI_15/XLXN_213/XLXI_15/XLXN_213_RSTF.LFBK));
```

```
GD_N <= ((NOT MF AND SD_N)
OR (SD_N AND NOT XLXN_123)
OR (MF AND TXE_N AND XLXN_123)
OR (XLXN_123 AND NOT XLXN_96 AND NOT XLXN_92.LFBK));
```

```
MBUS_I(0) <= NOT (((NOT ADCHB(0) AND NOT XLXN_131.LFBK AND NOT $OpTx$FX_DC$3.LFBK)
OR (NOT ADCLB(0) AND XLXN_131.LFBK AND NOT $OpTx$FX_DC$3.LFBK)
OR (NOT MBUS(0).PIN AND NOT XLXN_131.LFBK AND $OpTx$FX_DC$3.LFBK)));
MBUS(0) <= MBUS_I(0) when MBUS_OE(0) = '1' else 'Z';
MBUS_OE(0) <= (NOT MF AND XLXN_123 AND NOT $OpTx$FX_DC$11);
```

```
MBUS_I(1) <= NOT (((NOT ADCHB(1) AND NOT XLXN_131.LFBK AND NOT $OpTx$FX_DC$3.LFBK)
OR (NOT ADCLB(1) AND XLXN_131.LFBK AND NOT $OpTx$FX_DC$3.LFBK)
OR (NOT MBUS(1).PIN AND NOT XLXN_131.LFBK AND $OpTx$FX_DC$3.LFBK)));
MBUS(1) <= MBUS_I(1) when MBUS_OE(1) = '1' else 'Z';
MBUS_OE(1) <= (NOT MF AND XLXN_123 AND NOT $OpTx$FX_DC$11);
```

```
MBUS_I(2) <= MBUS(2);
MBUS(2) <= MBUS_I(2) when MBUS_OE(2) = '1' else 'Z';
MBUS_OE(2) <= (NOT MF AND XLXN_123 AND NOT $OpTx$FX_DC$11);
```

```

MBUS_I(3) <= NOT (((NOT ADCHB(3) AND NOT XLXN_131.LFBK AND NOT $Optx$FX_DC$3.LFBK)
OR (NOT ADCLB(3) AND XLXN_131.LFBK AND NOT $Optx$FX_DC$3.LFBK)
OR (NOT MBUS(3).PIN AND NOT XLXN_131.LFBK AND $Optx$FX_DC$3.LFBK)));
MBUS(3) <= MBUS_I(3) when MBUS_OE(3) = '1' else 'Z';
MBUS_OE(3) <= (NOT MF AND XLXN_123 AND NOT $Optx$FX_DC$11);

```

```

MBUS_I(4) <= UBUS(4);
MBUS(4) <= MBUS_I(4) when MBUS_OE(4) = '1' else 'Z';
MBUS_OE(4) <= (NOT MF AND XLXN_123 AND NOT $Optx$FX_DC$11);

```

```

MBUS_I(5) <= NOT (((NOT ADCHB(5) AND NOT XLXN_131.LFBK AND NOT $Optx$FX_DC$3.LFBK)
OR (NOT ADCLB(5) AND XLXN_131.LFBK AND NOT $Optx$FX_DC$3.LFBK)
OR (NOT MBUS(5).PIN AND NOT XLXN_131.LFBK AND $Optx$FX_DC$3.LFBK)));
MBUS(5) <= MBUS_I(5) when MBUS_OE(5) = '1' else 'Z';
MBUS_OE(5) <= (NOT MF AND XLXN_123 AND NOT $Optx$FX_DC$11);

```

```

MBUS_I(6) <= UBUS(6);
MBUS(6) <= MBUS_I(6) when MBUS_OE(6) = '1' else 'Z';
MBUS_OE(6) <= (NOT MF AND XLXN_123 AND NOT $Optx$FX_DC$11);

```

```

MBUS_I(7) <= UBUS(7);
MBUS(7) <= MBUS_I(7) when MBUS_OE(7) = '1' else 'Z';
MBUS_OE(7) <= (NOT MF AND XLXN_123 AND NOT $Optx$FX_DC$11.LFBK);

```

```

MCLR_N_I <= ((NOT STORE AND NOT XLXI_15/XLXN_248 AND NOT XLXN_96 AND
NOT XLXN_92.LFBK)
OR (NOT XLXN_123 AND NOT XLXI_15/XLXN_248 AND NOT XLXN_96 AND
NOT XLXN_92.LFBK)
OR (NOT XLXI_15/XLXN_248 AND NOT XLXN_96 AND
XLXI_15/XLXN_27.LFBK AND NOT XLXN_92.LFBK));
MCLR_N <= MCLR_N_I when MCLR_N_OE = '1' else 'Z';
MCLR_N_OE <= (XLXN_123 AND ML);

```

```

MD0 <= NOT PD0;

```

```

MD1 <= PD0
      XOR
MD1 <= PD1;

```

```

MD2 <= PD2
      XOR
MD2 <= (PD1 AND PD0);

```

```

MWRT_N_I <= NOT ((XLXN_123 AND XLXN_88.LFBK));
MWRT_N <= MWRT_N_I when MWRT_N_OE = '1' else 'Z';
MWRT_N_OE <= (XLXN_123 AND NOT $Optx$FX_DC$11);

```

```

OFF_I <= '0';
OFF <= OFF_I when OFF_OE = '1' else 'Z';

```



```

OFF_OE <= (XLXI_15/XLXN_239.LFBK AND $OpTx$FX_DC$4.LFBK);

SND <= ((XLXN_88 AND NOT XLXN_123.LFBK AND XLXN_149.LFBK)
        OR (MF AND NOT MWRT_N.PIN AND XLXN_123.LFBK AND XLXN_149.LFBK));

TCG <= (NOT XLXI_15/XLXN_27.LFBK AND $OpTx$FX_SC$9.LFBK);

UBUS_I(0) <= NOT (((XLXN_131 AND NOT ADCLB(0) AND NOT $OpTx$FX_DC$3)
                  OR (NOT XLXN_131 AND NOT ADCHB(0) AND NOT $OpTx$FX_DC$3)
                  OR (NOT XLXN_131 AND NOT MBUS(0).PIN AND $OpTx$FX_DC$3)));
UBUS(0) <= UBUS_I(0) when UBUS_OE(0) = '1' else 'Z';
UBUS_OE(0) <= RCLK;

UBUS_I(1) <= NOT (((XLXN_131 AND NOT ADCLB(1) AND NOT $OpTx$FX_DC$3)
                  OR (NOT XLXN_131 AND NOT ADCHB(1) AND NOT $OpTx$FX_DC$3)
                  OR (NOT XLXN_131 AND NOT MBUS(1).PIN AND $OpTx$FX_DC$3)));
UBUS(1) <= UBUS_I(1) when UBUS_OE(1) = '1' else 'Z';
UBUS_OE(1) <= RCLK;

UBUS_I(2) <= NOT (((XLXN_131 AND NOT ADCLB(2) AND NOT $OpTx$FX_DC$3)
                  OR (NOT XLXN_131 AND NOT ADCHB(2) AND NOT $OpTx$FX_DC$3)
                  OR (NOT XLXN_131 AND NOT MBUS(2).PIN AND $OpTx$FX_DC$3)));
UBUS(2) <= UBUS_I(2) when UBUS_OE(2) = '1' else 'Z';
UBUS_OE(2) <= RCLK;

UBUS_I(3) <= MBUS(3);
UBUS(3) <= UBUS_I(3) when UBUS_OE(3) = '1' else 'Z';
UBUS_OE(3) <= RCLK;

UBUS_I(4) <= NOT (((XLXN_131 AND NOT ADCLB(4) AND NOT $OpTx$FX_DC$3)
                  OR (NOT XLXN_131 AND NOT ADCHB(4) AND NOT $OpTx$FX_DC$3)
                  OR (NOT XLXN_131 AND NOT MBUS(4).PIN AND $OpTx$FX_DC$3)));
UBUS(4) <= UBUS_I(4) when UBUS_OE(4) = '1' else 'Z';
UBUS_OE(4) <= RCLK;

UBUS_I(5) <= MBUS(5);
UBUS(5) <= UBUS_I(5) when UBUS_OE(5) = '1' else 'Z';
UBUS_OE(5) <= RCLK;

UBUS_I(6) <= NOT (((XLXN_131 AND NOT ADCLB(6) AND NOT $OpTx$FX_DC$3)
                  OR (NOT XLXN_131 AND NOT ADCHB(6) AND NOT $OpTx$FX_DC$3)
                  OR (NOT XLXN_131 AND NOT MBUS(6).PIN AND $OpTx$FX_DC$3)));
UBUS(6) <= UBUS_I(6) when UBUS_OE(6) = '1' else 'Z';
UBUS_OE(6) <= RCLK;

UBUS_I(7) <= NOT (((XLXN_131 AND NOT ADCLB(7) AND NOT $OpTx$FX_DC$3)
                  OR (NOT XLXN_131 AND NOT ADCHB(7) AND NOT $OpTx$FX_DC$3)
                  OR (NOT XLXN_131 AND NOT MBUS(7).PIN AND $OpTx$FX_DC$3)));
UBUS(7) <= UBUS_I(7) when UBUS_OE(7) = '1' else 'Z';
UBUS_OE(7) <= RCLK;

```

```

WCG <= NOT ((MF AND NOT TXE_N AND NOT XLXN_92.LFBK AND NOT $Optx$FX_SC$9.LFBK)
OR (NOT TXE_N AND NOT XLXN_123 AND NOT XLXN_92.LFBK AND
NOT $Optx$FX_SC$9.LFBK)
OR (NOT MF AND NOT SD_N AND XLXN_123 AND NOT XLXN_92.LFBK AND
NOT $Optx$FX_SC$9.LFBK));

WRT <= ((XLXN_88 AND NOT XLXN_123.LFBK)
OR (MF AND NOT MWRT_N.PIN AND XLXN_123.LFBK));

FDCPE_XLXI_15/XLXN_213: FDCPE port map (XLXI_15/XLXN_213,UBUS(3).PIN,RCLK,NOT XLXI_1
XLXI_15/XLXN_213/XLXI_15/XLXN_213_RSTF <= ((XLXN_123 AND MCLR_N.PIN)
OR (NOT XLXN_92 AND NOT XLXN_123 AND NOT XLXI_15/XLXN_248 AND NOT XLXN_96));

FDCPE_XLXI_15/XLXN_239: FDCPE port map (XLXI_15/XLXN_239,UBUS(0).PIN,RCLK,'0','0');

FDCPE_XLXI_15/XLXN_248: FDCPE port map (XLXI_15/XLXN_248,STORE,NOT XLXI_15/XLXN_213/

FDCPE_XLXI_15/XLXN_27: FDCPE port map (XLXI_15/XLXN_27,XLXN_123,COIN,NOT XLXI_15/XLX
FDCPE_XLXI_15/XLXN_286: FDCPE port map (XLXI_15/XLXN_286,XLXI_15/XLXN_27.LFBK,NOT CO
FDCPE_XLXI_15/XLXN_42: FDCPE port map (XLXI_15/XLXN_42,XLXI_15/XLXN_27,NOT XLXI_15/X

XLXI_15/XLXN_42/XLXI_15/XLXN_42_CLKF <= (NOT STORE AND NOT REJ_N);

XLXI_15/XLXN_42/XLXI_15/XLXN_42_RSTF <= (XLXN_123 AND NOT XLXI_15/XLXN_248 AND XLXI_
MCLR_N.PIN);

FDCPE_XLXI_15/XLXN_49: FDCPE port map (XLXI_15/XLXN_49,XLXI_15/XLXN_27.LFBK,XLXI_15/
XLXI_15/XLXN_49_C <= (PSB AND NOT XLXI_15/XLXN_286.LFBK);

FDCPE_XLXI_3/XLXN_506: FDCPE port map (XLXI_3/XLXN_506,XLXN_88.LFBK,WCLK,XLXI_3/XLXN
XLXI_3/XLXN_506_CLR <= (NOT XLXI_3/XLXN_595 AND XLXI_3/XLXN_506.LFBK);

FDCPE_XLXI_3/XLXN_595: FDCPE port map (XLXI_3/XLXN_595,NOT WCG,WCLK,WCG,'0');

FTCPE_XLXI_4/XLXN_143: FTCPE port map (XLXI_4/XLXN_143,'1',NOT XLXI_4/XLXN_35,XLXI_4
XLXI_4/XLXN_143_CLR <= (NOT XLXI_15/XLXN_27 AND $Optx$FX_SC$9);

FTCPE_XLXI_4/XLXN_275: FTCPE port map (XLXI_4/XLXN_275,'1',NOT XLXI_4/XLXN_143.LFBK,
XLXI_4/XLXN_275_CLR <= (NOT XLXI_15/XLXN_27 AND $Optx$FX_SC$9);

FTCPE_XLXI_4/XLXN_277: FTCPE port map (XLXI_4/XLXN_277,'1',NOT XLXI_4/XLXN_275.LFBK,
XLXI_4/XLXN_277_CLR <= (NOT XLXI_15/XLXN_27 AND $Optx$FX_SC$9);

FTCPE_XLXI_4/XLXN_295: FTCPE port map (XLXI_4/XLXN_295,'1',NOT XLXI_4/XLXN_85,XLXI_4
XLXI_4/XLXN_295_CLR <= (NOT XLXI_15/XLXN_27 AND $Optx$FX_SC$9);

FTCPE_XLXI_4/XLXN_296: FTCPE port map (XLXI_4/XLXN_296,'1',NOT XLXI_4/XLXN_295.LFBK,
XLXI_4/XLXN_296_CLR <= (NOT XLXI_15/XLXN_27 AND $Optx$FX_SC$9);

FTCPE_XLXI_4/XLXN_298: FTCPE port map (XLXI_4/XLXN_298,'1',NOT XLXI_4/XLXN_299,XLXI_
XLXI_4/XLXN_298_CLR <= (NOT XLXI_15/XLXN_27 AND $Optx$FX_SC$9);

```

```
FTCPE_XLXI_4/XLXN_299: FTCPE port map (XLXI_4/XLXN_299,'1',NOT XLXI_4/XLXN_296,XLXI_
XLXI_4/XLXN_299_CLR <= (NOT XLXI_15/XLXN_27 AND $Optx$FX_SC$9);
```

```
FTCPE_XLXI_4/XLXN_35: FTCPE port map (XLXI_4/XLXN_35,'1',NOT XLXI_4/XLXN_78.LFBK,XLX
XLXI_4/XLXN_35_CLR <= (NOT XLXI_15/XLXN_27 AND $Optx$FX_SC$9);
```

```
FTCPE_XLXI_4/XLXN_78: FTCPE port map (XLXI_4/XLXN_78,'1',NOT XLXI_4/XLXN_79.LFBK,XLX
XLXI_4/XLXN_78_CLR <= (NOT XLXI_15/XLXN_27 AND $Optx$FX_SC$9);
```

```
FTCPE_XLXI_4/XLXN_79: FTCPE port map (XLXI_4/XLXN_79,'1',TCLK,XLXI_4/XLXN_79_CLR,'0'
XLXI_4/XLXN_79_CLR <= (NOT XLXI_15/XLXN_27 AND $Optx$FX_SC$9);
```

```
FTCPE_XLXI_4/XLXN_85: FTCPE port map (XLXI_4/XLXN_85,'1',NOT XLXI_4/XLXN_277,XLXI_4/
XLXI_4/XLXN_85_CLR <= (NOT XLXI_15/XLXN_27 AND $Optx$FX_SC$9);
```

```
FDCPE_XLXN_123: FDCPE port map (XLXN_123,UBUS(2).PIN,RCLK,'0','0');
```

```
FDCPE_XLXN_130: FDCPE port map (XLXN_130,XLXN_131.LFBK,XLXI_3/XLXN_595,XLXN_130_CLR,
XLXN_130_CLR <= ($Optx$FX_SC$9 AND XLXN_130.LFBK);
```

```
FDCPE_XLXN_131: FDCPE port map (XLXN_131,XLXN_131_D,XLXI_3/XLXN_595,XLXN_131_CLR,'0'
XLXN_131_D <= (NOT XLXN_92 AND $Optx$FX_DC$11);
XLXN_131_CLR <= ($Optx$FX_SC$9 AND XLXN_131.LFBK);
```

```
FDCPE_XLXN_149: FDCPE port map (XLXN_149,UBUS(1).PIN,RCLK,'0','0');
```

```
FDCPE_XLXN_88: FDCPE port map (XLXN_88,XLXN_88_D,WCLK,XLXN_88_CLR,'0');
XLXN_88_D <= (XLXI_3/XLXN_595 AND NOT XLXN_92.LFBK AND NOT XLXN_88.LFBK AND
NOT XLXI_3/XLXN_506.LFBK);
XLXN_88_CLR <= (NOT XLXI_3/XLXN_595 AND XLXN_88.LFBK);
```

```
FDCPE_XLXN_92: FDCPE port map (XLXN_92,XLXN_130,XLXI_3/XLXN_595,XLXN_92_CLR,'0');
XLXN_92_CLR <= (XLXN_92.LFBK AND $Optx$FX_SC$9.LFBK);
```

```
FDCPE_XLXN_96: FDCPE port map (XLXN_96,'1',NOT XLXI_4/XLXN_298,XLXN_96_CLR,'0');
XLXN_96_CLR <= (NOT XLXI_15/XLXN_27 AND $Optx$FX_SC$9);
```

```
YF <= (MF AND NOT XLXN_123);
```

```
YL <= (NOT XLXN_123 AND ML);
```

Register Legend:

```
FDCPE (Q,D,C,CLR,PRE);
FTCPE (Q,D,C,CLR,PRE);
LDCP (Q,D,G,CLR,PRE);
```

```
□***** Device Pin Out *****
```

```
Device : XC9572-15-PC84
```

```
-----
/11 10 9 8 7 6 5 4 3 2 1 84 83 82 81 80 79 78 77 76 75 \
| 12                                     74 |
| 13                                     73 |
| 14                                     72 |
| 15                                     71 |
```

16		70
17		69
18		68
19		67
20		66
21	XC9572-15-PC84	65
22		64
23		63
24		62
25		61
26		60
27		59
28		58
29		57
30		56
31		55
32		54
\ 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 /		

Pin Signal
No. Name

1 YF
2 ML
3 GD_N
4 MWRT_N
5 MCLR_N
6 SD_N
7 PGND
8 GND
9 WCLK
10 TCLK
11 WCG
12 RCLK
13 TCG
14 YL
15 MF
16 GND
17 PD2
18 PD1
19 PD0
20 TXE_N
21 WRT
22 VCC
23 UBUS<7>
24 UBUS<6>
25 UBUS<5>
26 UBUS<4>
27 GND
28 TDI
29 TMS
30 TCK
31 UBUS<3>
32 UBUS<2>
33 UBUS<1>
34 UBUS<0>
35 SND
36 PGND
37 PGND

Pin Signal
No. Name

43 ADCLB<0>
44 ADCLB<1>
45 ADCLB<2>
46 ADCLB<3>
47 ADCLB<4>
48 ADCLB<5>
49 GND
50 PGND
51 ADCLB<6>
52 ADCLB<7>
53 ADCHB<0>
54 ADCHB<1>
55 ADCHB<2>
56 ADCHB<3>
57 ADCHB<4>
58 OFF
59 TDO
60 GND
61 STORE
62 PSB
63 CLEAR
64 VCC
65 REJ_N
66 MBUS<7>
67 MBUS<6>
68 MBUS<5>
69 MBUS<4>
70 MBUS<3>
71 MBUS<2>
72 MBUS<1>
73 VCC
74 MBUS<0>
75 PGND
76 PGND
77 COIN
78 VCC
79 ADCHB<5>

38 VCC	80 ADCHB<6>
39 PGND	81 ADCHB<7>
40 PGND	82 MD0
41 PGND	83 MD1
42 GND	84 MD2

Legend : NC = Not Connected, unbonded pin
 PGND = Unused I/O configured as additional Ground pin
 TIE = Unused I/O floating -- must tie to VCC, GND or other signal
 VCC = Dedicated Power Pin
 GND = Dedicated Ground Pin
 TDI = Test Data In, JTAG pin
 TDO = Test Data Out, JTAG pin
 TCK = Test Clock, JTAG pin
 TMS = Test Mode Select, JTAG pin
 PROHIBITED = User reserved pin

□***** Compiler Options *****

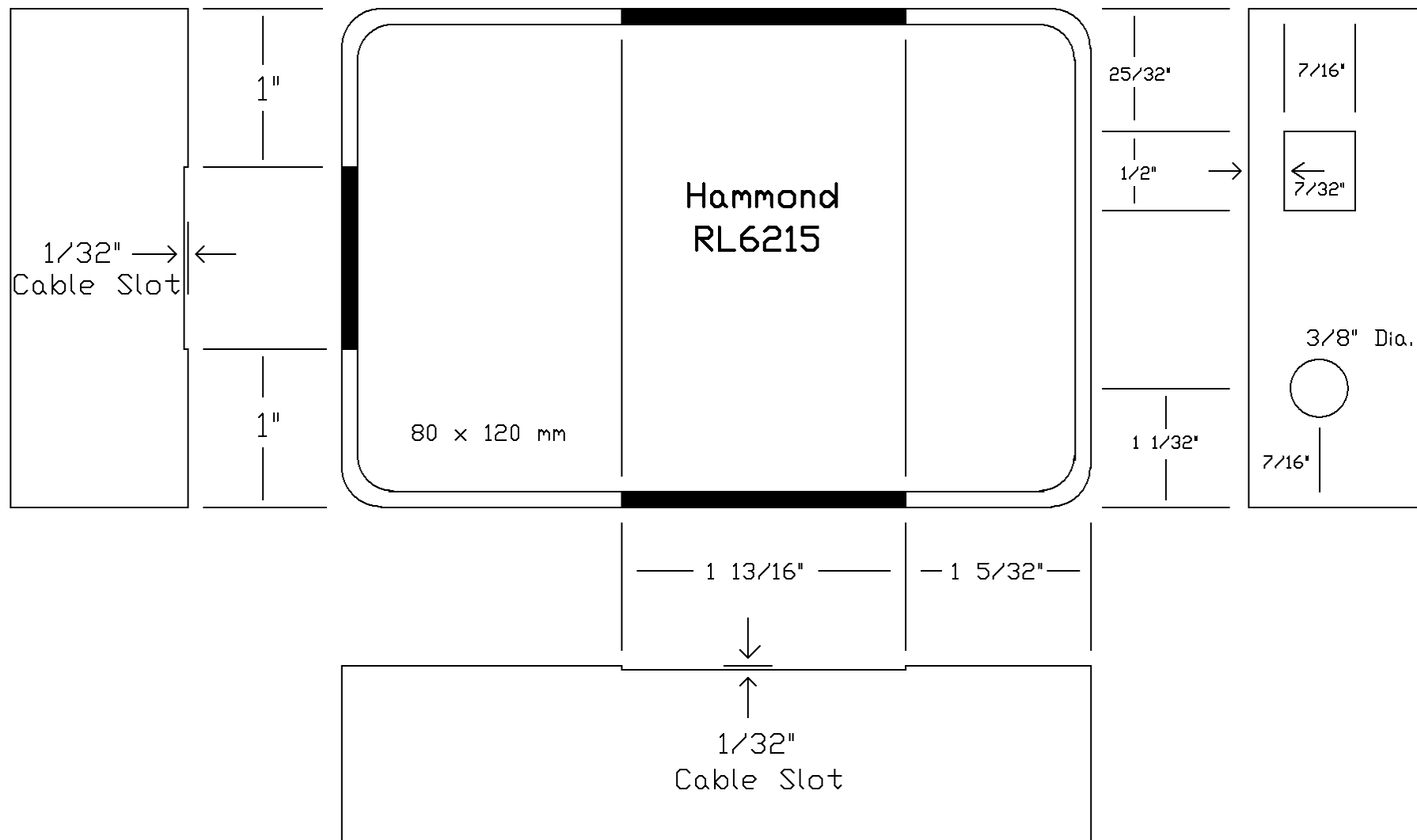
Following is a list of all global compiler options used by the fitter run.

Device(s) Specified	: xc9572-15-PC84
Optimization Method	: DENSITY
Multi-Level Logic Optimization	: ON
Ignore Timing Specifications	: OFF
Default Register Power Up Value	: LOW
Keep User Location Constraints	: ON
What-You-See-Is-What-You-Get	: OFF
Exhaustive Fitting	: OFF
Keep Unused Inputs	: OFF
Slew Rate	: SLOW
Power Mode	: STD
Ground on Unused IOs	: ON
Global Clock Optimization	: ON
Global Set/Reset Optimization	: ON
Global Ouput Enable Optimization	: ON
FASTConnect/UIM optimzation	: ON
Local Feedback	: ON
Pin Feedback	: ON
Input Limit	: 36
Pterm Limit	: 25

[back to top](#)

[print page](#)

Multiparameter IO Module - ADC





DLP-USB245M-G USB to FIFO Parallel Interface Module

The DLP-USB245M-G is the 2nd generation, lead free version of DLP Design's USB adapter. This device adds extra functionality to it's DLP-USB1 predecessor with a reduced component count and a new low price.

DLP-USB245M FEATURES

- Send / Receive Data over USB at up to 1 M Bytes / sec
- 384 byte FIFO Transmit buffer / 128 byte FIFO receive buffer for high data throughput
- Simple interface to CPU or MCU bus
- No in-depth knowledge of USB required as all USB
- Protocol is handled automatically within the module
- FTDI's Virtual COM port drivers eliminate the need for USB driver development in most cases.
- Integrated 3.3v Regulator – No External Regulator Required
- Integrated Power-On-Reset circuit
- Integrated 6MHz – 48Mhz clock multiplier PLL
- USB Bulk or Isocronous data transfer modes
- 4.4v - 5.25v Single Supply taken directly from the USB port
- UHCI / OHCI / EHCI host controller compatible
- USB 1.1 and USB 2.0 compatible
- USB VID, PID, Serial Number and Product Description
- Strings stored in on-board EEPROM.
- EEPROM programmable on-board via USB
- VIRTUAL COM PORT (VCP) DRIVERS for
 - Windows 98 and Windows 98 SE
 - Windows 2000 / ME / XP
 - Windows CE **
 - MAC OS-8 and OS-9
 - MAC OS-X
 - Linux 2.40 and greater
- D2XX (USB Direct Drivers + DLL S/W Interface)
 - Windows 98 and Windows 98 SE
 - Windows 2000 / ME / XP

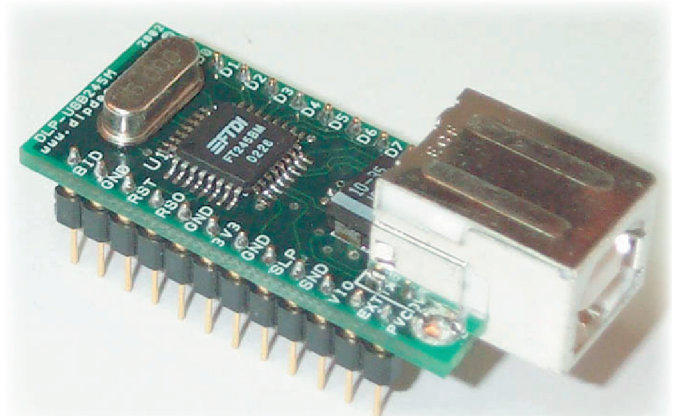
[** = In planning or under development]

APPLICATION AREAS

- Rapid Prototype Development
- USB ISDN and ADSL Modems
- High Speed USB PDA Interface
- USB Interface for Digital Cameras
- USB Interface for MP3 players
- High Speed USB Instrumentation
- USB Smart Card Readers
- Set Top Box (S.T.B.) PC-USB interface
- USB Hardware Modems
- USB Wireless Modems
- USB Bar Code Readers

GENERAL DESCRIPTION

The DLP-USB245M provides an easy cost-effective method of transferring data to / from a peripheral and a host at up to 8 Million bits (1-Megabyte) per second. Its simple FIFO-like design makes it easy to interface to any microcontroller or microprocessor via IO ports.



To send data from the peripheral to the host computer simply write the byte wide data into the module when TXE# is low. If the (384 byte) transmit buffer fills up or is busy storing the previously written byte, the device takes TXE# high in order to stop further data from being written until some of the FIFO data has been transferred over USB to the host.

When the host sends data to the peripheral over USB, the device will take RXF# low to let the peripheral know that at least one byte of data is available. The peripheral then reads the data until RXF# goes high indicating no more data is available to read.

By using FTDI's virtual COM Port drivers, the peripheral looks like a standard COM Port to the application software. Commands to set the baud rate are ignored – the device always transfers data at its fastest rate regardless of the application's baud rate setting. The latest versions of the drivers are available for download from DLP Design's website at <http://www.dlpdesign.com>.

APPLICATION NOTES

USB devices transfer data in packets. If data is to be sent from the PC, a packet is built up by the application program and is sent via the device driver to the USB scheduler. This scheduler puts a request onto the list of tasks for the USB host controller to perform. This will typically take at least 1 millisecond to execute because it will not pick up the new request until the next 'USB Frame' (the frame period is 1 millisecond).

There is therefore a sizeable overhead (depending on your required throughput) associated with moving the data from the application to the USB device. If data is sent 'byte at a time' by an application, this will severely limit the overall throughput of the system as a whole.

It must be stressed that in order to achieve maximum throughput, application programs should send or receive data using buffers and not individual characters.

DRIVER SOFTWARE

FTDI's VCP (Virtual COM Port) driver executable files are provided royalty-free on the condition that they are only used with designs incorporating an FTDI device (ie the FT245BM and DLP-USB245M). The latest version of the drivers can be downloaded from dlpdesign.com or ftdichip.com.

The VCP driver download file is a combined set of drivers for Windows '98, Windows ME and Windows 2000/XP. Unzip the file to a blank floppy disk or folder on your PC. The drivers can co-exist on the same floppy disk or folder since the INF files determine which set of drivers to load for each operating system version. Once loaded, the VCP drivers allow your application software, running on the host PC, to communicate with the DLP-USB245M as though it were connected to a COM (RS-232) port.

In addition to VCP drivers, FTDI's "D2XX Direct Drivers" for Windows offer an alternative solution to the VCP drivers that allows application software to interface with the FT245BM device using a DLL instead of a Virtual Com Port. The architecture of the D2XX drivers consists of a Windows WDM driver that communicates with the FT245BM device via the Windows USB Stack and a DLL that interfaces the Application Software (written in VC++, C++ Builder, Delphi, VB etc...) to the WDM driver. An INF installation file, uninstaller program and D2XX programmers guide complete the package

The D2XX Direct Drivers add support for simultaneous access and control of multiple FT245BM devices. The Extended Open function (FT_OpenEx) allows the device to be opened either by it's Product Description or by Serial Number, both of which can be programmed to be unique. The List Devices function (FT-ListDevices) allows the application software to determine which devices are currently available for use, again by product description or by serial number.

Several new features have been recently added to the D2XX drivers. Functions are provided to program the EEPROM (FT_EE_Program), and read the EEPROM (FT_EE_Read). Unused space in the EEPROM is called the User Area (EEUA), and functions are provided to access the EEUA. FT_EE_UASize gets its size, FT_EE_UAWrite writes data into it, and FT_EE_UARead is used to read its contents.

Download FTDI Application Notes AN232-03, AN232-05, AN232-06, and AN232-07 for detailed instructions on how to install and remove the drivers.

EEPROM WRITE UTILITY

The DLP-USB245M has the option to accept manufacturer-specific information that is written into the 93C46 EEPROM. Parameters that can be programmed include the VID and the PID identifiers, the manufacturer's product string and a serial number.

FTD2XXST is the latest EEPROM serializer and testing utility for the FT245BM device. It replaces the VCP based 232Prog utility. FTD2XXST is based on the new D2XX drivers and will work on Win '98, Win ME and Win 2000/XP platforms. You must install the latest release of the D2XX drivers in order to run this application. Refer to the SER200.PDF guide that is included in the EEPROM Write Utility's download zip file for details on how to use the serializer.

If you have VCP drivers installed on the PC that is to perform the EEPROM write process you must uninstall these drivers using the uninstaller program (included with the driver files) and install the D2XX drivers prior to running the serializer utility.

QUICK START GUIDE

This guide requires the use of a Windows 98/2000 PC that is equipped with a USB port.

1. Download the DLL version of the device drivers from either dlpdesign.com or ftdichip.com. Unzip the drivers onto a blank floppy disk or into a folder on the hard drive.
2. Download the serializer program from either dlpdesign.com or ftdichip.com. Unzip the package and place it in a folder on the hard drive.
3. Select a power source for the DLP-USB245M module via pins 10, 11, and 12. Be sure to pull the RESET# pin high (pin 3).
4. Connect the DLP-USB245M board to the PC via a standard, 6-foot USB cable. This action initiates the loading of the USB drivers. When prompted, select the folder where the DLL version of the device drivers were stored

in step one. Windows will then complete the installation of the device drivers for the DLP-USB245M board. The next time the DLP-USB245M module is attached, the host PC will immediately load the correct drivers without any prompting. Reboot the PC if prompted to do so.

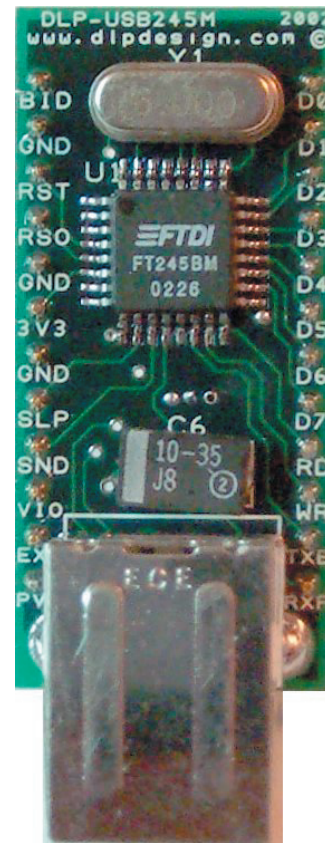
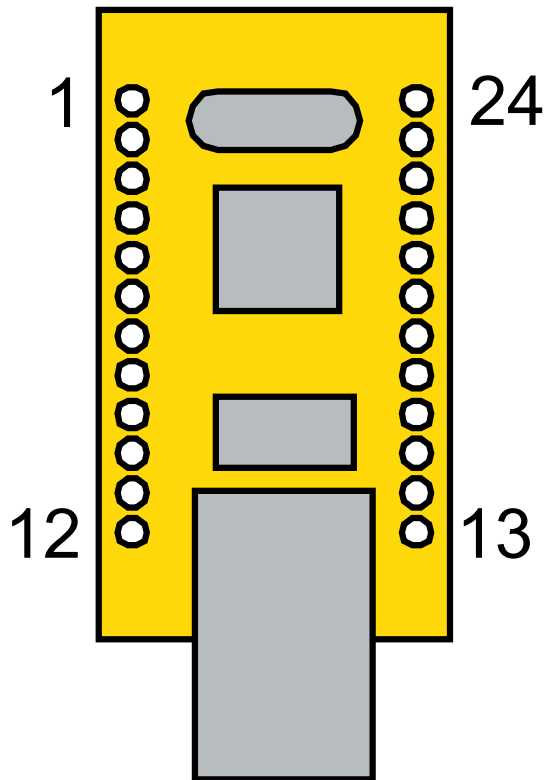
5. Run the serializer program and write the VID (0403), PID (6001), a description string of your choosing and manufacturers ID as instructed in the instruction manual that was downloaded with the serializer software. Terminate the serializer program and disconnect the DLP-USB245M board from the USB cable. Wait 10 seconds and reconnect the DLP-USB245M board. Reboot the PC if instructed to do so.

NOTE 1 - The DLP-USB245M comes pre-programmed with a default VID and PID. Step 5 is optional and only required if different VID/PID codes are required.

NOTE 2 – The “Test” button on the serializer program is intended for use with the DLP-USB232M module (and FT232BM device) and will fail if used with the DLP-USB245M module. The “Read” function will work and will present the contents of the EEPROM as well as the serial number and description strings stored in the EEPROM.

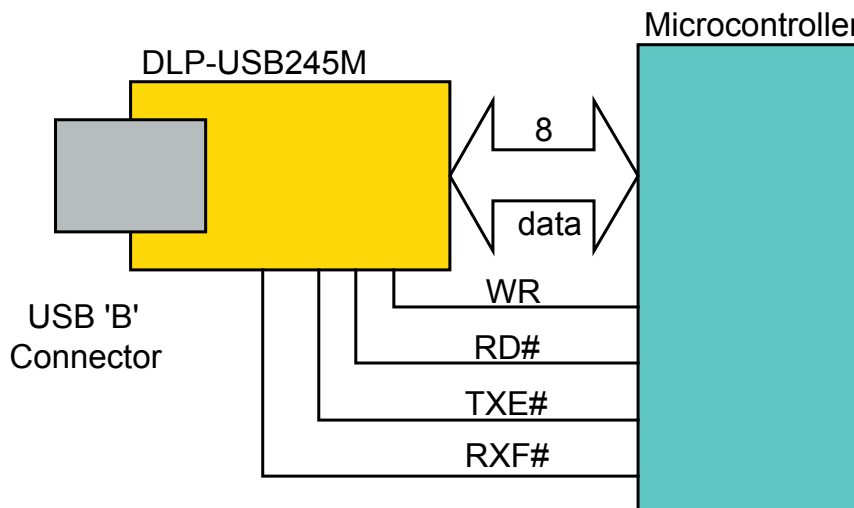
At this point the DLP-USB245M is ready for use. Note that the DLP-USB245M will appear non-responsive if data sent from the host PC is not read from the FT245BM device by an attached microcontroller/microprocessor/DSP/FPGA/ etc...

Table 1 - DLP-USB245M PINOUT DESCRIPTION



Pin#	Description
1	BOARD ID (Out) Identifies the board as either a DLP-USB245M or DLP-USB232M. High for DLP-USB232M and low for DLP-USB245M.
2	Ground
3	RESET# (In) Can be used by an external device to reset the FT245BM. If not required this pin must be tied to VCC.
4	RESETO# (Out) Output of the internal Reset Generator. Stays high impedance for ~ 2ms after VCC > 3.5v and the internal clock starts up, then clamps it's output to the 3.3v output of the internal regulator. Taking RESET# low will also force RSTOUT# to go high impedance. RSTOUT# is NOT affected by a USB Bus Reset.
5	Ground
6	3V3OUT (Out) Output from the integrated L.D.O. regulator. It's primary purpose is to provide the internal 3.3v supply to the USB transceiver cell and the RSTOUT# pin. A small amount of current (<= 5mA) can be drawn from this pin to power external 3.3v logic if required.
7	Ground
8	SLEEP (Out) Goes Low after the device is configured via USB, then high during USB suspend. Can be used to control power to external logic using a P-Channel Logic Level MOSFET switch.

9	SND/WUP (In) If the DLP-USB245M is in USB suspend, a positive edge on this pin (WAKEUP) initiates a remote wakeup sequence. If the device is active (not in suspend) a positive edge on this pin (SEND) causes the data in the write buffer to be sent to the PC on the next USB Data-In request regardless of how many bytes are in the buffer.
10	VCC-IO (In) 3.0 volt to +5.25 volt VCC to the UART interface pins 10..12, 14..16 and 18..25. When interfacing with 3.3v external logic connect VCC-IO to the 3.3v supply of the external logic, otherwise connect to VCC to drive out at 5v CMOS level. This pin must be connected to VCC from the target electronics or EXTVCC.
11	EXTVCC – (In) Use for applying main power (4.4 to 5.25 Volts) to the module. Connect to PORTVCC if module is to be powered by the USB port (typical configuration)
12	PORTVCC - (Out) Power from USB port. Connect to EXTVCC if module is to be powered by the USB port (typical configuration). 500mA maximum current available to USB adapter and target electronics if USB device is configured for high power.
13	RXF# - (Out) When low, at least 1 byte is present in the FIFO's 128-byte receive buffer and is ready to be read with RD#. RXF# goes high when the receive buffer is empty.
14	TXE# - When high, the FIFO's 385 byte transmit buffer is full, or busy storing the last byte written. Do not attempt to write data to the transmit buffer when TXE# is high.
15	WR (In) When taken from a high to a low state, WR reads the 8 data lines and writes the byte into the FIFO's transmit buffer. Data written to the transmit buffer is sent to the host PC within the TX buffer timeout value (default 16mS) and placed in the RS-232 buffer opened by the application program. Note : The FT245BM allows the TX buffer timeout value to be reprogrammed to a value between 1 and 255mS depending on the applicaton requirement, also the SND pin can be used to send any remaining data in the TX buffer regardless of the timeout value.
16	RD# (In) When pulled low, RD# takes the 8 data lines from a high impedance state to the current byte in the FIFO's receive buffer. Taking RD# high returns the data pins to a high impedance state and prepares the next byte (if available) in the FIFO to be read.
17	D7 I/O Bi-directional Data Bus Bit # 7
18	D6 I/O Bi-directional Data Bus Bit # 6
19	D5 I/O Bi-directional Data Bus Bit # 5
20	D4 I/O Bi-directional Data Bus Bit # 4
21	D3 I/O Bi-directional Data Bus Bit # 3
22	D2 I/O Bi-directional Data Bus Bit # 2
23	D1 I/O Bi-directional Data Bus Bit # 1
24	D0 I/O Bi-directional Data Bus Bit # 0



Basic Connection to a Microcontroller

USB Bus Powered and Self Powered Configuration

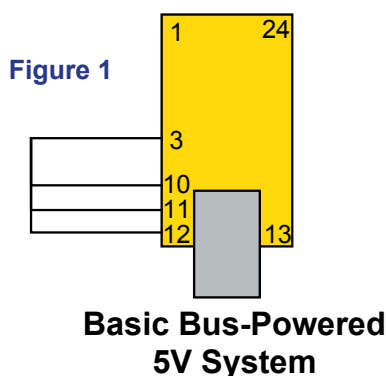


Figure 1 illustrates a typical USB bus powered configuration. A USB Bus Powered device gets its power from the USB bus. Basic rules for USB Bus power devices are as follows –

- a) On plug-in, the device must draw no more than 100mA
- b) On USB Suspend the device must draw no more than 500uA.
- c) A Bus Powered High Power Device (one that draws more than 100mA) should use the SLEEP# pin to keep the current below 100mA on plug-in and 500uA on USB suspend.
- d) A device that consumes more than 100mA can not be plugged into a USB

Bus Powered Hub

e) No device can draw more that 500mA from the USB Bus.

f) The power descriptor in the EEPROM should be programmed to a value of zero.

Figure 2 illustrates a typical USB self powered configuration. A USB Self Powered device gets its power from its own Power Supply and does not draw current from the USB bus.

Basic rules for USB Self power devices are as follows –

- a) A Self-Powered device should not force current down the USB bus when the USB Host or Hub Controller is powered down.
- b) A Self Powered Device can take as much current as it likes during normal operation and USB suspend as it has its own Power Supply.
- c) A Self Powered Device can be used with any USB Host and both Bus and Self Powered USB Hubs.

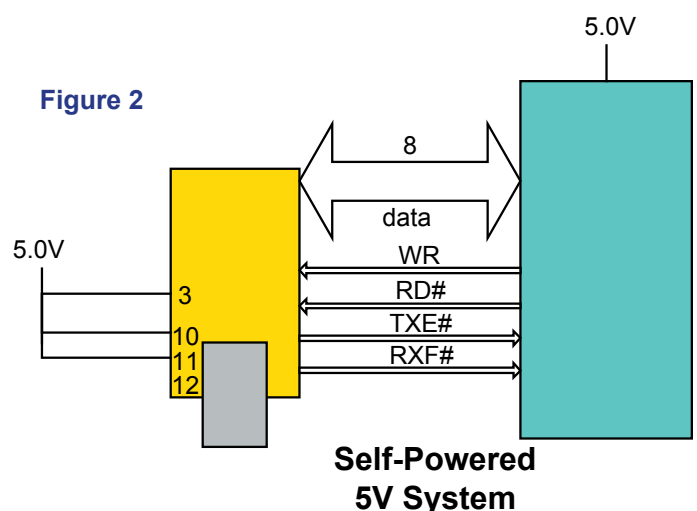


Figure 3

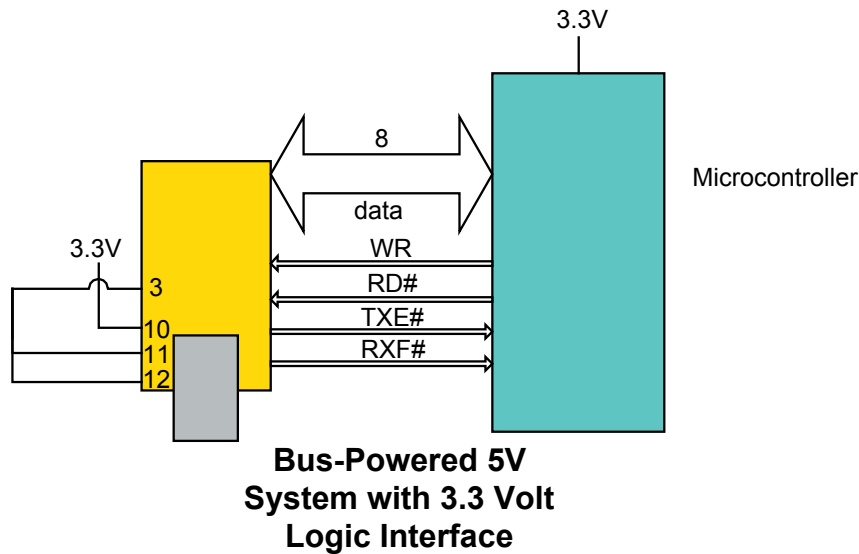


Figure 3 shows how to configure the DLP-USB245M to interface with a 3.3v logic device. In this example, the target electronics provides the 3.3 volts via the VCCIO line (pin 10) which in turn will cause the FIFO interface IO pins to drive out at 3.3v level.

Bus Powered Circuit with Power Control

USB Bus powered circuits need to be able to power down in USB suspend mode in order to meet the $\leq 500\mu\text{A}$ total suspend current requirement (including external logic).

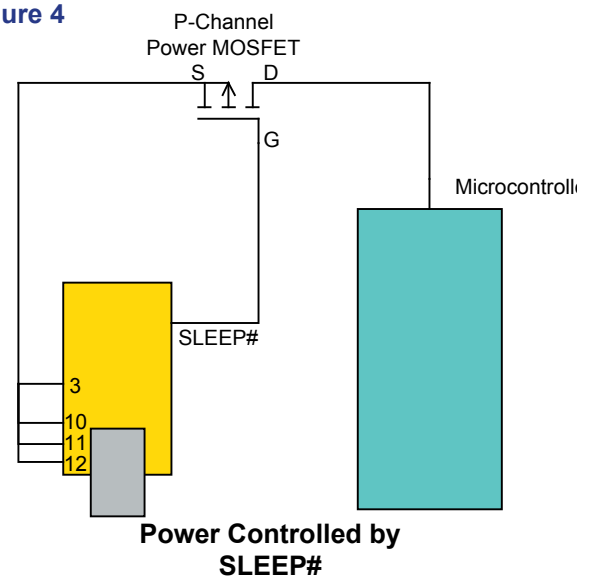
Figure 4 shows how to use a discrete P-Channel Logic Level MOSFET to control the power to external logic circuits. A suitable device could be a Fairchild NDT456P or equivalent. This configuration is suitable for powering external logic where the normal supply current is $\leq 100\text{mA}$ and the logic to be controlled does not generate an appreciable current surge at power-up. For power switching external logic that takes over 100mA or generates a current surge on powerup we

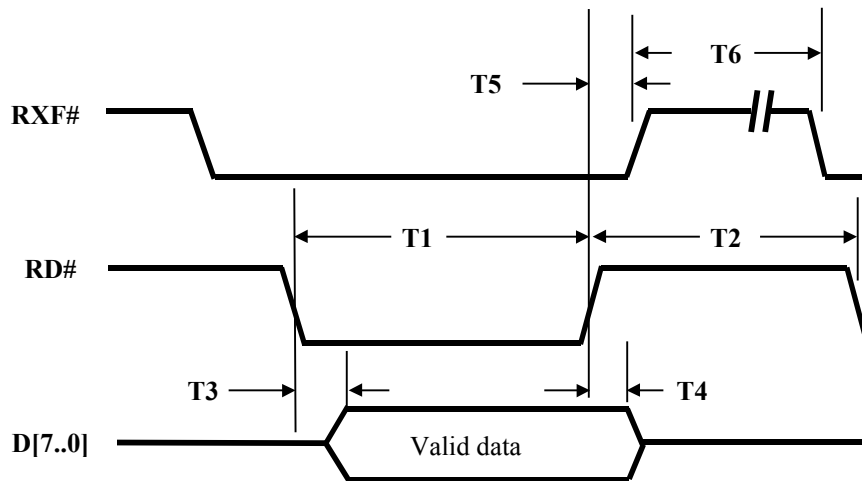
recommend that a dedicated power switch i.c with inbuilt “soft-start” is used instead of a MOSFET. A suitable power switch i.c. for such an application would be a Micrel (www.micrel.com) MIC2025-2BM or equivalent.

Please note the following points in connection with power controlled designs –

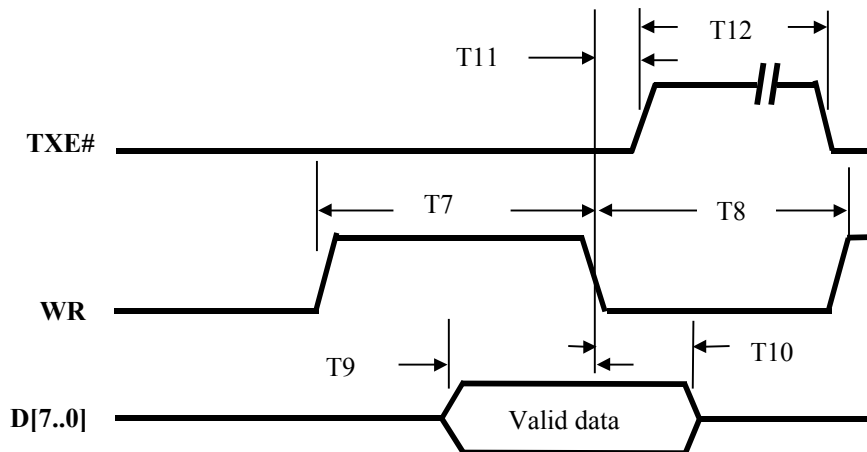
- The logic to be controlled must have it's own reset circuitry so that it will automatically reset itself when power is re-applied on coming out of suspend.
- Set the soft pull-down option bit in the FT232BM EEPROM.
- For 3.3v power controlled circuits VCCIO must not be powered down with the external circuitry (PWREN# gets it's VCC supply from VCCIO).

Figure 4



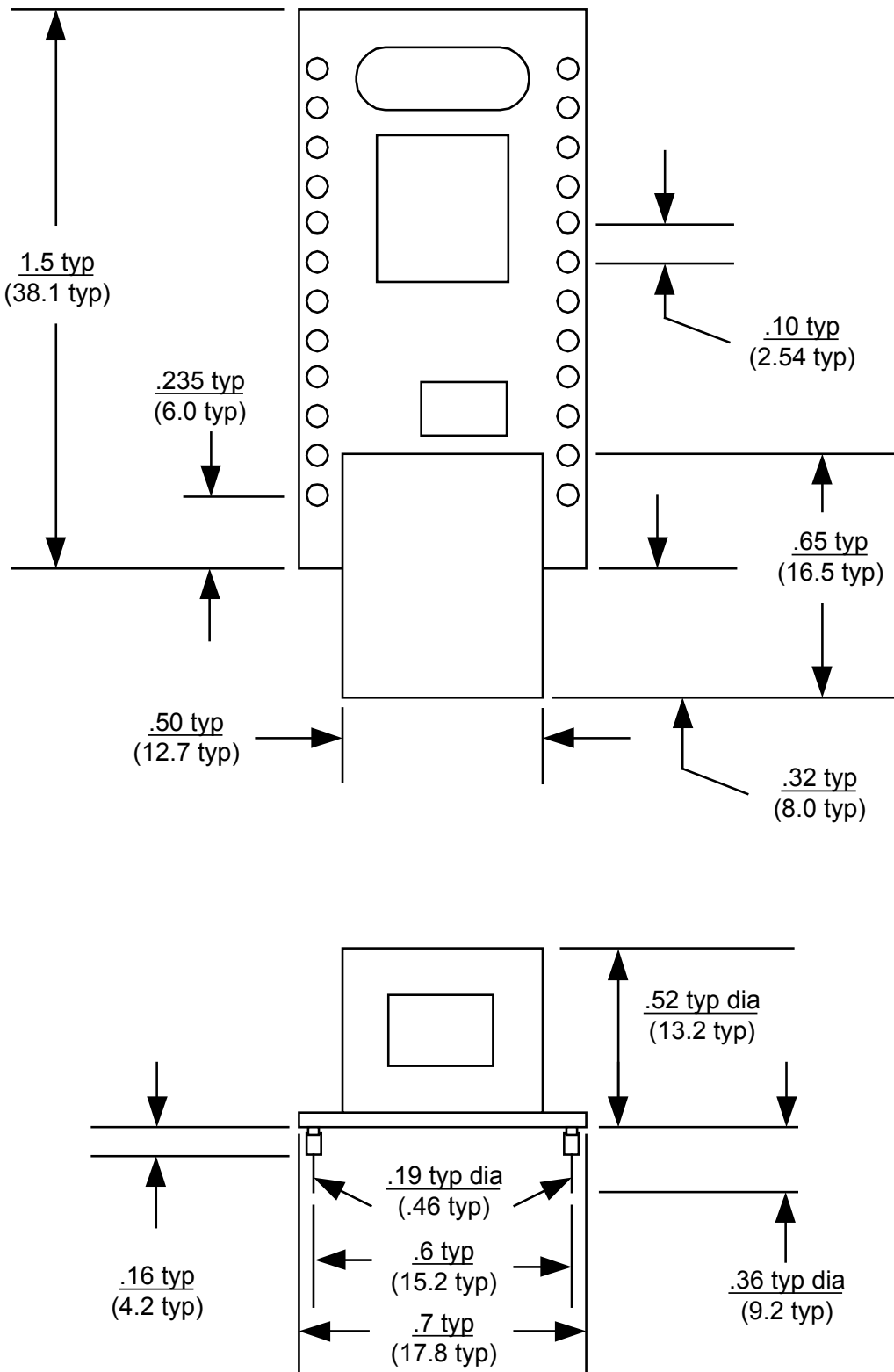
DLP-USB245M TIMING DIAGRAM – FIFO READ CYCLE

Time	Description	Min	Max	Unit
T1	RD Active Pulse Width	50		ns
T2	RD to RD Pre-Charge Time	50		ns
T3	RD Active to Valid Data		30	ns
T4	Valid Data Hold Time from RD Inactive	10		ns
T5	RD Inactive to RXF#	5	25	ns
T6	RXF inactive after RD cycle	80		ns

DLP-USB1 TIMING DIAGRAM – FIFO WRITE CYCLE

Time	Description	Min	Max	Unit
T7	WR Active Pulse Width	50		ns
T8	WR to WR Pre-Charge Time	50		ns
T9	Data Setup Time before WR inactive		20	ns
T10	Data Hold Time from WR inactive	10		ns
T11	WR Inactive to TXE#	5	25	ns
T12	TXE inactive after RD cycle	80		Ns

Mechanical Drawings (PRELIMINARY)
Inches(millimeters) unless otherwise noted



Absolute Maximum Ratings

These are the absolute maximum ratings for the FT245BM device in accordance with the Absolute Maximum Rating System (IEC 60134). Exceeding these may cause permanent damage to the device.

- Storage Temperature -65°C to + 150°C
- Ambient Temperature (Power Applied)..... 0°C to + 70°C
- VCC Supply Voltage -0.5v to +6.00v
- DC Input Voltage - Inputs -0.5v to VCC + 0.5v
- DC Input Voltage - High Impedance Bidirectionals -0.5v to VCC + 0.5v
- DC Output Current – Outputs 24mA
- DC Output Current – Low Impedance Bidirectionals 24mA
- Power Dissipation (VCC = 5.25v) 500mW
- Electrostatic Discharge Voltage (I < 1uA) +/- 2000v
- Latch Up Current (Vi < 0 or Vi > Vcc) 100mA

DC Characteristics (Ambient Temperature = 0 .. 70°C)

Operating Voltage and Current

Parameter	Description	Min	Typ	Max	Units	Conditions
Vcc1	VCC Operating Supply Voltage	4.4	5.0	5.25	V	
Vcc2	VCCIO Operating Supply Voltage	3.0	-	5.25	V	
Icc1	Operating Supply Current	-	25	-	mA	Normal Operation
Icc2	Operating Supply Current	-	350	400	uA	USB Suspend

FIFO IO Pin Characteristics (VCCIO = 5.0v)

Parameter	Description	Min	Typ	Max	Units	Conditions
Voh	Output Voltage High	4.4	-	4.9	V	I source = 2mA
Vol	Output Voltage Low	0.1	-	0.7	V	I sink = 4 mA
Vin	Input Switching Threshold	1.1	1.5	1.9	V	Note 1
VHys	Input Switching Hysteresis		200		mV	

FIFO IO Pin Characteristics (VCCIO = 3.3v)

Parameter	Description	Min	Typ	Max	Units	Conditions
Voh	Output Voltage High	2.7	-	3.2	V	I source = 2mA
Vol	Output Voltage Low	0.1	-	0.7	V	I sink = 4 mA
Vin	Input Switching Threshold	1.0	1.4	1.8	V	Note 1
VHys	Input Switching Hysteresis		200		mV	

RESET# Pin Characteristics

Parameter	Description	Min	Typ	Max	Units	Conditions
V_{in}	Input Switching Threshold	1.1	1.5	1.9	V	Note 1
V_{Hys}	Input Switching Hysteresis		200		mV	

Note 1 – Inputs have an internal 200k pull-up resistor to VCCIO.

RSTOUT Pin Characteristics

Parameter	Description	Min	Typ	Max	Units	Conditions
V_{oh}	Output Voltage High	3.0	-	3.6	V	I source = 2mA
I_{ol}	Leakage Current Tri-State	-	-	5	uA	

USB IO Pin Characteristics

Parameter	Description	Min	Typ	Max	Units	Conditions
UV_{oh}	IO Pins Static Output (High)	2.8		3.6v	V	RI = 1k5 to 3V3Out (D+) RI = 15k to GND (D-)
UV_{ol}	IO Pins Static Output (Low)	0		0.3	V	RI = 1k5 to 3V3Out (D+) RI = 15k to GND (D-)
UV_{se}	Single Ended Rx Threshold	0.8		2.0	V	
UCom	Differential Common Mode	0.8		2.5	V	
UV_{Dif}	Differential Input Sensitivity	0.2			V	
UDrvZ	Driver Output Impedance	29		44	ohm	Note 2

Note 2 – Driver Output Impedance includes the external 27R series resistors on USBDP and USBDM pins.

Disclaimer

Neither the whole nor any part of the information contained in, or the product described in this datasheet, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder.

This product and its documentation are supplied on an as is basis and no warranty as to their suitability for any particular purpose is either made or implied. DLP Design will not accept any claim for damages howsoever arising as a result of use or failure of this product. Your statutory rights are not affected.

This product or any variant of it is not intended for use in any medical appliance, device or system in which the failure of the product might reasonably be expected to result in personal injury.

This document provides preliminary information that may be subject to change without notice.

Contact Information

DLP Design

1605 Roma Lane

Allen, TX 75013

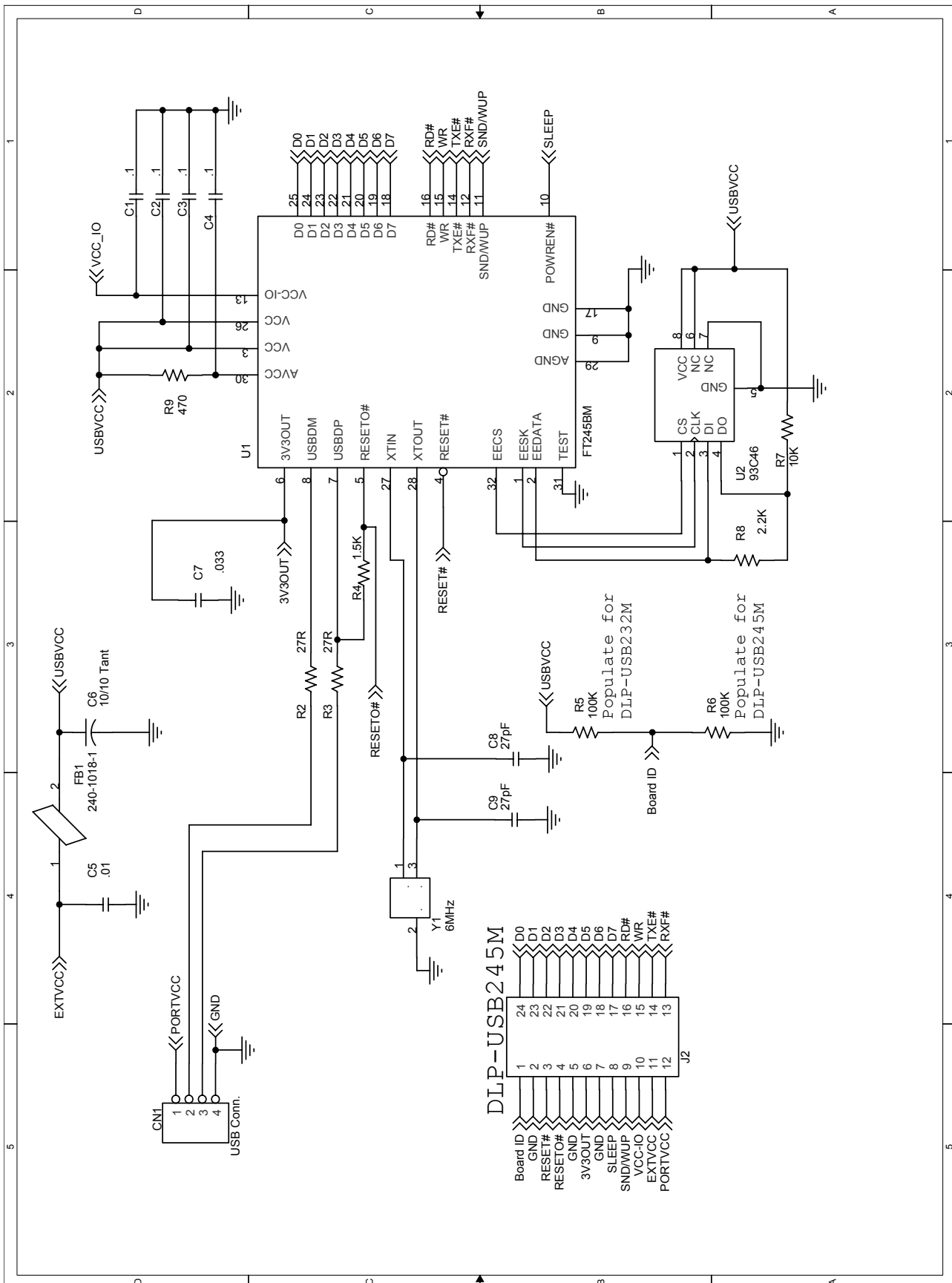
Phone: 469-964-8027

Fax: 415-901-4859

Email: support@dlpdesign.com

Internet: <http://www.dlpdesign.com>

Appendix A – DLP USB245M Schematic





Future Technology Devices International Ltd.

Software Application Development D2XX Programmer's Guide

Document Reference No.: FT_000071

Version 1.2

Issue Date: 2011-04-25

FTDI provides DLL and virtual COM port (VCP) application interfaces to its drivers. This document provides the application programming interface (API) for the FTD2XX DLL function library.

Future Technology Devices International Ltd (FTDI)

Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow, G41 1HH, United Kingdom

Tel.: +44 (0) 141 429 2777 Fax: + 44 (0) 141 429 2758

E-Mail (Support): support1@ftdichip.com

Web: www.ftdichip.com

Use of FTDI devices in life support and/or safety applications is entirely at the user's risk, and the user agrees to defend, indemnify and hold harmless FTDI from any and all damages, claims, suits or expense resulting from such use.

Table of Contents

1	Preface	4
1.1	Acronyms and Abbreviations	4
2	Introduction.....	5
3	D2XX Classic Functions.....	6
3.1	FT_SetVIDPID.....	6
3.2	FT_GetVIDPID	6
3.3	FT_CreateDeviceInfoList	7
3.4	FT_GetDeviceInfoList.....	8
3.5	FT_GetDeviceInfoDetail	9
3.6	FT_ListDevices	11
3.7	FT_Open.....	13
3.8	FT_OpenEx.....	14
3.9	FT_Close	16
3.10	FT_Read	17
3.11	FT_Write	19
3.12	FT_SetBaudRate.....	20
3.13	FT_SetDivisor	21
3.14	FT_SetDataCharacteristics	21
3.15	FT_SetTimeouts.....	22
3.16	FT_SetFlowControl.....	23
3.17	FT_SetDtr	24
3.18	FT_ClrDtr	25
3.19	FT_SetRts.....	26
3.20	FT_ClrRts	27
3.21	FT_GetModemStatus.....	28
3.22	FT_GetQueueStatus	29
3.23	FT_GetDeviceInfo	30
3.24	FT_GetDriverVersion.....	31
3.25	FT_GetLibraryVersion	32
3.26	FT_GetComPortNumber	33
3.27	FT_GetStatus	34
3.28	FT_SetEventNotification	34
3.29	FT_SetChars	37

3.30	FT_SetBreakOn.....	39
3.31	FT_SetBreakOff	39
3.32	FT_Purge.....	40
3.33	FT_ResetDevice.....	41
3.34	FT_ResetPort	42
3.35	FT_CyclePort	42
3.36	FT_Rescan	43
3.37	FT_Reload	44
3.38	FT_SetResetPipeRetryCount.....	45
3.39	FT_StopInTask.....	46
3.40	FT_RestartInTask	47
3.41	FT_SetDeadmanTimeout.....	48
3.42	FT_IoCtl	49
3.43	FT_SetWaitMask.....	49
3.44	FT_WaitOnMask.....	49
4	EEPROM Programming Interface Functions	50
4.1	FT_ReadEE.....	50
4.2	FT_WriteEE	50
4.3	FT_EraseEE.....	51
4.4	FT_EE_Read.....	51
4.5	FT_EE_ReadEx	53
4.6	FT_EE_Program.....	54
4.7	FT_EE_ProgramEx	57
4.8	FT_EE_UASize	58
4.9	FT_EE_UARead	59
4.10	FT_EE_UAWrite	61
5	Extended API Functions	63
5.1	FT_SetLatencyTimer	63
5.2	FT_GetLatencyTimer.....	64
5.3	FT_SetBitMode	65
5.4	FT_GetBitMode.....	66
5.5	FT_SetUSBParameters	67
6	FT-Win32 API Functions	69
6.1	FT_W32_CreateFile	69

6.2	FT_W32_CloseHandle	71
6.3	FT_W32_ReadFile.....	71
6.4	FT_W32_WriteFile.....	74
6.5	FT_W32_GetOverlappedResult	76
6.6	FT_W32_EscapeCommFunction	76
6.7	FT_W32_GetCommModemStatus	77
6.8	FT_W32_SetupComm.....	78
6.9	FT_W32_SetCommState	79
6.10	FT_W32_GetCommState	80
6.11	FT_W32_SetCommTimeouts	80
6.12	FT_W32_GetCommTimeouts.....	82
6.13	FT_W32_SetCommBreak	82
6.14	FT_W32_ClearCommBreak.....	83
6.15	FT_W32_SetCommMask	84
6.16	FT_W32_GetCommMask.....	85
6.17	FT_W32_WaitCommEvent	86
6.18	FT_W32_PurgeComm	88
6.19	FT_W32_GetLastError.....	89
6.20	FT_W32_ClearCommError	89
7	Contact Information.....	92
8	Appendix A - Type Definitions	94
9	Appendix B - Revision History	102

1 Preface

The D2XX interface is a proprietary interface specifically for FTDI devices. This document provides an explanation of the functions available to application developers via the FTD2XX library

Any software code examples given in this document are for information only. The examples are not guaranteed and are not supported by FTDI.

1.1 Acronyms and Abbreviations

Terms	Description
CDM	Combined Driver Model. Windows driver package which incorporates both D2XX and VCP drivers.
D2XX	FTDI's proprietary "direct" driver interface via FTD2XX.DLL
VCP	Virtual COM Port

Table 1.1 Acronyms and Abbreviations

2 Introduction

FTDI provides two alternative software interfaces for its range of USB-UART and USB-FIFO ICs. One interface provides a Virtual COM Port (VCP) which appears to the system as a legacy COM port. The second interface, D2XX, is provided via a proprietary DLL (FTD2XX.DLL). The D2XX interface provides special functions that are not available in standard operating system COM port APIs, such as setting the device into a different mode or writing data into the device EEPROM.

In the case of the FTDI drivers for Windows, the D2XX driver and VCP driver are distributed in the same driver package, called the Combined Driver Model (CDM) package. Figure 2.1 Windows CDM Driver Architecture illustrates the architecture of the Windows CDM driver.

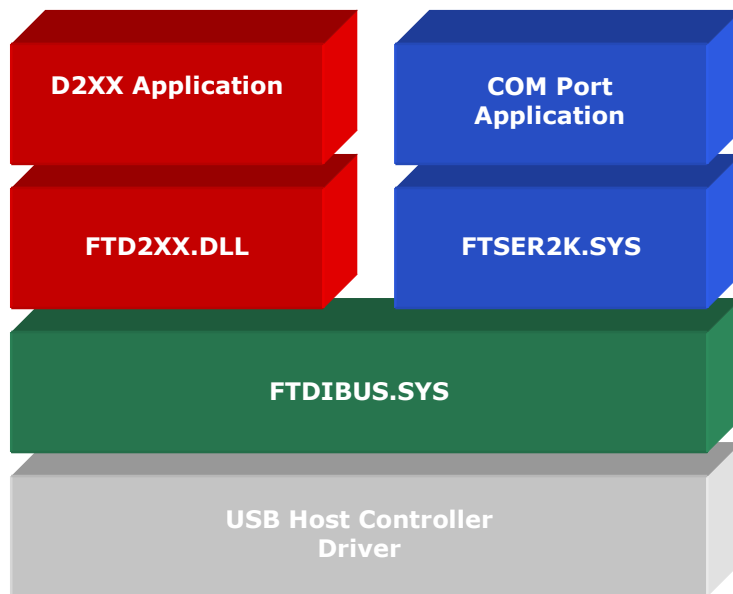


Figure 2.1 Windows CDM Driver Architecture

For Linux, Mac OS X (10.4 and later) and Windows CE (4.2 and later) the D2XX driver and VCP driver are mutually exclusive options as only one driver type may be installed at a given time for a given device ID. In the case of a Windows system running the CDM driver, applications may use either the D2XX or VCP interface without installing a different driver but may not use both interfaces at the same time.

As the VCP driver interface is designed to emulate a legacy COM port, FTDI does not provide documentation on how to communicate with the VCP driver from an application; the developer is referred to the large amount of material available on the Internet regarding serial communication.

The D2XX interface is a proprietary interface specifically for FTDI devices. This document provides an explanation of the functions available to application developers via the FTD2XX library.

3 D2XX Classic Functions

The functions listed in this section are compatible with all FTDI devices.

3.1 FT_SetVIDPID

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Summary

A command to include a custom VID and PID combination within the internal device list table. This will allow the driver to load for the specified VID and PID combination.

Definition

FT_STATUS **FT_SetVIDPID** (DWORD *dwVID*, DWORD *dwPID*)

Parameters

dwVID Device Vendor ID (VID)

dwPID Device Product ID (PID)

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

By default, the driver will support a limited set of VID and PID matched devices (VID 0x0403 with PIDs 0x6001, 0x6010, 0x6006 only).

In order to use the driver with other VID and PID combinations the FT_SetVIDPID function must be used prior to calling [FT_ListDevices](#), [FT_Open](#), [FT_OpenEx](#) or [FT_CreateDeviceInfoList](#).

3.2 FT_GetVIDPID

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Summary

A command to retrieve the current VID and PID combination from within the internal device list table.

Definition

FT_STATUS **FT_GetVIDPID** (DWORD * *pdwVID*, DWORD * *pdwPID*)

Parameters

pdwVID Pointer to DWORD that will contain the internal VID

pdwPID Pointer to DWORD that will contain the internal PID

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

See [FT_SetVIDPID](#).

3.3 FT_CreateDeviceInfoList

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function builds a device information list and returns the number of D2XX devices connected to the system. The list contains information about both unopen and open devices.

Definition

FT_STATUS **FT_CreateDeviceInfoList** (LPDWORD *lpdwNumDevs*)

Parameters

<i>lpdwNumDevs</i>	Pointer to unsigned long to store the number of devices connected.
--------------------	--

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

An application can use this function to get the number of devices attached to the system. It can then allocate space for the device information list and retrieve the list using [FT_GetDeviceInfoList](#) or [FT_GetDeviceInfoDetail](#).

If the devices connected to the system change, the device info list will not be updated until FT_CreateDeviceInfoList is called again.

Example

```
FT_STATUS ftStatus;  
DWORD numDevs;  
  
// create the device information list  
ftStatus = FT_CreateDeviceInfoList(&numDevs);  
if (ftStatus == FT_OK) {  
    printf("Number of devices is %d\n", numDevs);  
}  
else {  
    // FT_CreateDeviceInfoList failed  
}
```

3.4 FT_GetDeviceInfoList

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function returns a device information list and the number of D2XX devices in the list.

Definition

FT_STATUS **FT_GetDeviceInfoList** (FT_DEVICE_LIST_INFO_NODE *pDest,
LPDWORD lpdwNumDevs)

Parameters

*pDest	Pointer to an array of FT_DEVICE_LIST_INFO_NODE structures.
lpdwNumDevs	Pointer to the number of elements in the array.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function should only be called after calling [FT_CreateDeviceInfoList](#). If the devices connected to the system change, the device info list will not be updated until [FT_CreateDeviceInfoList](#) is called again.

Location ID information is not returned for devices that are open when [FT_CreateDeviceInfoList](#) is called.

Information is not available for devices which are open in other processes. In this case, the *Flags* parameter of the [FT_DEVICE_LIST_INFO_NODE](#) will indicate that the device is open, but other fields will be unpopulated.

The flag value is a 4-byte bit map containing miscellaneous data as defined [Appendix A – Type Definitions](#). Bit 0 (least significant bit) of this number indicates if the port is open (1) or closed (0). Bit 1 indicates if the device is enumerated as a high-speed USB device (2) or a full-speed USB device (0). The remaining bits (2 - 31) are reserved.

The array of [FT_DEVICE_LIST_INFO_NODES](#) contains all available data on each device. The structure of [FT_DEVICE_LIST_INFO_NODES](#) is given in the Appendix. The storage for the list must be allocated by the application. The number of devices returned by [FT_CreateDeviceInfoList](#) can be used to do this.

When programming in Visual Basic, LabVIEW or similar languages, [FT_GetDeviceInfoDetail](#) may be required instead of this function.

Please note that Linux, Mac OS X and Windows CE do not support location IDs. As such, the Location ID parameter in the structure will be empty under these operating systems.

Example

```
FT_STATUS ftStatus;
FT_DEVICE_LIST_INFO_NODE *devInfo;
DWORD numDevs;

// create the device information list
ftStatus = FT_CreateDeviceInfoList(&numDevs);

if (ftStatus == FT_OK) {
    printf("Number of devices is %d\n", numDevs);
}

if (numDevs > 0) {
    // allocate storage for list based on numDevs
    devInfo =
    (FT_DEVICE_LIST_INFO_NODE*)malloc(sizeof(FT_DEVICE_LIST_INFO_NODE)*numDevs);
    // get the device information list
    ftStatus = FT_GetDeviceInfoList(devInfo, &numDevs);
    if (ftStatus == FT_OK) {
        for (int i = 0; i < numDevs; i++) {
            printf("Dev %d:\n", i);
            printf("  Flags=0x%x\n", devInfo[i].Flags);
            printf("  Type=0x%x\n", devInfo[i].Type);
            printf("  ID=0x%x\n", devInfo[i].ID);
            printf("  LocId=0x%x\n", devInfo[i].LocId);
            printf("  SerialNumber=%s\n", devInfo[i].SerialNumber);
            printf("  Description=%s\n", devInfo[i].Description);
            printf("  ftHandle=0x%x\n", devInfo[i].ftHandle);
        }
    }
}
```

3.5 FT_GetDeviceInfoDetail

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function returns an entry from the device information list.

Definition

FT_STATUS **FT_GetDeviceInfoDetail** (DWORD *dwIndex*, LPDWORD *lpdwFlags*,
LPDWORD *lpdwType*,
LPDWORD *lpdwID*, LPDWORD *lpdwLocId*,
PCHAR *pcSerialNumber*, PCHAR *pcDescription*,
FT_HANDLE **ftHandle*)

Parameters

<i>dwIndex</i>	Index of the entry in the device info list.
<i>lpdwFlags</i>	Pointer to unsigned long to store the flag value.
<i>lpdwType</i>	Pointer to unsigned long to store device type.

<i>lpdwID</i>	Pointer to unsigned long to store device ID.
<i>lpdwLocId</i>	Pointer to unsigned long to store the device location ID.
<i>pcSerialNumber</i>	Pointer to buffer to store device serial number as a null-terminated string.
<i>pcDescription</i>	Pointer to buffer to store device description as a null-terminated string.
<i>*ftHandle</i>	Pointer to a variable of type FT_HANDLE where the handle will be stored.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function should only be called after calling [FT_CreateDeviceInfoList](#). If the devices connected to the system change, the device info list will not be updated until [FT_CreateDeviceInfoList](#) is called again.

The index value is zero-based.

The flag value is a 4-byte bit map containing miscellaneous data as defined [Appendix A – Type Definitions](#). Bit 0 (least significant bit) of this number indicates if the port is open (1) or closed (0). Bit 1 indicates if the device is enumerated as a high-speed USB device (2) or a full-speed USB device (0). The remaining bits (2 - 31) are reserved.

Location ID information is not returned for devices that are open when [FT_CreateDeviceInfoList](#) is called.

Information is not available for devices which are open in other processes. In this case, the *lpdwFlags* parameter will indicate that the device is open, but other fields will be unpopulated.

To return the whole device info list as an array of [FT_DEVICE_LIST_INFO_NODE](#) structures, use [FT_CreateDeviceInfoList](#).

Please note that Linux, Mac OS X and Windows CE do not support location IDs. As such, the Location ID parameter in the structure will be empty under these operating systems.

Example

```
FT_STATUS ftStatus;
FT_HANDLE ftHandleTemp;
DWORD numDevs;
DWORD Flags;
DWORD ID;
DWORD Type;
DWORD LocId;
char SerialNumber[16];
char Description[64];

// create the device information list
ftStatus = FT_CreateDeviceInfoList(&numDevs);
if (ftStatus == FT_OK) {
    printf("Number of devices is %d\n", numDevs);
}

if (numDevs > 0) {
    // get information for device 0
    ftStatus = FT_GetDeviceInfoDetail(0, &Flags, &Type, &ID, &LocId, SerialNumber,
    Description, &ftHandleTemp);
    if (ftStatus == FT_OK) {
        printf("Dev 0:\n");
        printf("  Flags=0x%x\n", Flags);
        printf("  Type=0x%x\n", Type);
    }
}
```



```
printf(" ID=0x%x\n", ID);  
printf(" LocId=0x%x\n", LocId);  
printf(" SerialNumber=%s\n", SerialNumber);  
printf(" Description=%s\n", Description);  
printf(" ftHandle=0x%x\n", ftHandleTemp);  
}  
}
```

3.6 FT_ListDevices

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Gets information concerning the devices currently connected. This function can return information such as the number of devices connected, the device serial number and device description strings, and the location IDs of connected devices.

Definition

FT_STATUS **FT_ListDevices** (PVOID *pvArg1*, PVOID *pvArg2*, DWORD *dwFlags*)

Parameters

<i>pvArg1</i>	Meaning depends on <i>dwFlags</i> .
<i>pvArg2</i>	Meaning depends on <i>dwFlags</i> .
<i>dwFlags</i>	Determines format of returned information.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function can be used in a number of ways to return different types of information. A more powerful way to get device information is to use the [FT_CreateDeviceInfoList](#), [FT_GetDeviceInfoList](#) and [FT_GetDeviceInfoDetail](#) functions as they return all the available information on devices.

In its simplest form, it can be used to return the number of devices currently connected. If [FT_LIST_NUMBER_ONLY](#) bit is set in *dwFlags*, the parameter *pvArg1* is interpreted as a pointer to a DWORD location to store the number of devices currently connected.

It can be used to return device information: if [FT_OPEN_BY_SERIAL_NUMBER](#) bit is set in *dwFlags*, the serial number string will be returned; if [FT_OPEN_BY_DESCRIPTION](#) bit is set in *dwFlags*, the product description string will be returned; if [FT_OPEN_BY_LOCATION](#) bit is set in *dwFlags*, the Location ID will be returned; if none of these bits is set, the serial number string will be returned by default.

It can be used to return device string information for a single device. If [FT_LIST_BY_INDEX](#) and [FT_OPEN_BY_SERIAL_NUMBER](#) or [FT_OPEN_BY_DESCRIPTION](#) bits are set in *dwFlags*, the parameter *pvArg1* is interpreted as the index of the device, and the parameter *pvArg2* is interpreted as a pointer to

a buffer to contain the appropriate string. Indexes are zero-based, and the error code [FT_DEVICE_NOT_FOUND](#) is returned for an invalid index.

It can be used to return device string information for all connected devices. If [FT_LIST_ALL](#) and [FT_OPEN_BY_SERIAL_NUMBER](#) or [FT_OPEN_BY_DESCRIPTION](#) bits are set in *dwFlags*, the parameter *pvArg1* is interpreted as a pointer to an array of pointers to buffers to contain the appropriate strings and the parameter *pvArg2* is interpreted as a pointer to a DWORD location to store the number of devices currently connected. Note that, for *pvArg1*, the last entry in the array of pointers to buffers should be a NULL pointer so the array will contain one more location than the number of devices connected.

The location ID of a device is returned if [FT_LIST_BY_INDEX](#) and [FT_OPEN_BY_LOCATION](#) bits are set in *dwFlags*. In this case the parameter *pvArg1* is interpreted as the index of the device, and the parameter *pvArg2* is interpreted as a pointer to a variable of type long to contain the location ID. Indexes are zero-based, and the error code [FT_DEVICE_NOT_FOUND](#) is returned for an invalid index. Please note that Windows CE and Linux do not support location IDs.

The location IDs of all connected devices are returned if [FT_LIST_ALL](#) and [FT_OPEN_BY_LOCATION](#) bits are set in *dwFlags*. In this case, the parameter *pvArg1* is interpreted as a pointer to an array of variables of type long to contain the location IDs, and the parameter *pvArg2* is interpreted as a pointer to a DWORD location to store the number of devices currently connected.

Examples

The examples that follow use these variables.

```
FT_STATUS ftStatus;  
DWORD numDevs;
```

1. Get the number of devices currently connected

```
ftStatus = FT_ListDevices(&numDevs,NULL,FT_LIST_NUMBER_ONLY);  
if (ftStatus == FT_OK) {  
    // FT_ListDevices OK, number of devices connected is in numDevs  
}  
else {  
    // FT_ListDevices failed  
}
```

2. Get serial number of first device

```
DWORD devIndex = 0; // first device  
char Buffer[64]; // more than enough room!  
  
ftStatus =  
FT_ListDevices((PVOID)devIndex,Buffer,FT_LIST_BY_INDEX|FT_OPEN_BY_SERIAL_NUMBER);  
if (ftStatus == FT_OK) {  
    // FT_ListDevices OK, serial number is in Buffer  
}  
else {  
    // FT_ListDevices failed  
}
```

Note that indexes are zero-based. If more than one device is connected, incrementing *devIndex* will get the serial number of each connected device in turn.

3. Get device descriptions of all devices currently connected

```
char *BufPtrs[3];           // pointer to array of 3 pointers  
char Buffer1[64];           // buffer for description of first device  
char Buffer2[64];           // buffer for description of second device  
  
// initialize the array of pointers  
BufPtrs[0] = Buffer1;  
BufPtrs[1] = Buffer2;  
BufPtrs[2] = NULL;         // last entry should be NULL
```

```
ftStatus = FT_ListDevices (BufPtrs, &numDevs, FT_LIST_ALL|FT_OPEN_BY_DESCRIPTION);
if (ftStatus == FT_OK) {
    // FT_ListDevices OK, product descriptions are in Buffer1 and Buffer2, and
    // numDevs contains the number of devices connected
}
else {
    // FT_ListDevices failed
}
```

Note that this example assumes that two devices are connected. If more devices are connected, then the size of the array of pointers must be increased and more description buffers allocated.

4. Get locations of all devices currently connected

```
long locIdBuf[16];

ftStatus = FT_ListDevices (locIdBuf, &numDevs, FT_LIST_ALL|FT_OPEN_BY_LOCATION);
if (ftStatus == FT_OK) {
    // FT_ListDevices OK, location IDs are in locIdBuf, and
    // numDevs contains the number of devices connected
}
else {
    // FT_ListDevices failed
}
```

Note that this example assumes that no more than 16 devices are connected. If more devices are connected, then the size of the array of pointers must be increased.

3.7 FT_Open

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Open the device and return a handle which will be used for subsequent accesses.

Definition

FT_STATUS **FT_Open** (int *iDevice*, FT_HANDLE **ftHandle*)

Parameters

<i>iDevice</i>	Index of the device to open. Indices are 0 based.
<i>ftHandle</i>	Pointer to a variable of type FT_HANDLE where the handle will be stored. This handle must be used to access the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

Although this function can be used to open multiple devices by setting *iDevice* to 0, 1, 2 etc. there is no ability to open a specific device. To open named devices, use the function [FT_OpenEx](#).

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0,&ftHandle);
if (ftStatus == FT_OK) {
    // FT_Open OK, use ftHandle to access device
}
else {
    // FT_Open failed
}
```

3.8 FT_OpenEx

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Open the specified device and return a handle that will be used for subsequent accesses. The device can be specified by its serial number, device description or location.

This function can also be used to open multiple devices simultaneously. Multiple devices can be specified by serial number, device description or location ID (location information derived from the physical location of a device on USB). Location IDs for specific USB ports can be obtained using the utility USBView and are given in hexadecimal format. Location IDs for devices connected to a system can be obtained by calling [FT_GetDeviceInfoList](#) or [FT_ListDevices](#) with the appropriate flags.

Definition

FT_STATUS **FT_OpenEx** (PVOID *pvArg1*, DWORD *dwFlags*, FT_HANDLE **ftHandle*)

Parameters

<i>pvArg1</i>	Pointer to an argument whose type depends on the value of <i>dwFlags</i> . It is normally be interpreted as a pointer to a null terminated string.
<i>dwFlags</i>	FT_OPEN_BY_SERIAL_NUMBER , FT_OPEN_BY_DESCRIPTION or FT_OPEN_BY_LOCATION .
<i>ftHandle</i>	Pointer to a variable of type FT_HANDLE where the handle will be stored. This handle must be used to access the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

The parameter specified in *pvArg1* depends on *dwFlags*: if *dwFlags* is [FT_OPEN_BY_SERIAL_NUMBER](#), *pvArg1* is interpreted as a pointer to a null-terminated string that represents the serial number of the device; if *dwFlags* is [FT_OPEN_BY_DESCRIPTION](#), *pvArg1* is interpreted as a pointer to a null-terminated string that represents the device description; if *dwFlags* is [FT_OPEN_BY_LOCATION](#), *pvArg1* is interpreted as a long value that contains the location ID of the device. Please note that Windows CE and Linux do not support location IDs.

ftHandle is a pointer to a variable of type *FT_HANDLE* where the handle is to be stored. This handle must be used to access the device.

Examples

The examples that follow use these variables.

```
FT_STATUS ftStatus;  
FT_STATUS ftStatus2;  
FT_HANDLE ftHandle1;  
FT_HANDLE ftHandle2;  
long dwLoc;
```

1. Open a device with serial number "FT000001"

```
ftStatus = FT_OpenEx("FT000001", FT_OPEN_BY_SERIAL_NUMBER, &ftHandle1);  
if (ftStatus == FT_OK) {  
    // success - device with serial number "FT000001" is open  
}  
else {  
    // failure  
}
```

2. Open a device with device description "USB Serial Converter"

```
ftStatus = FT_OpenEx("USB Serial Converter", FT_OPEN_BY_DESCRIPTION, &ftHandle1);  
if (ftStatus == FT_OK) {  
    // success - device with device description "USB Serial Converter" is open  
}  
else {  
    // failure  
}
```

3. Open 2 devices with serial numbers "FT000001" and "FT999999"

```
ftStatus = FT_OpenEx("FT000001", FT_OPEN_BY_SERIAL_NUMBER, &ftHandle1);  
ftStatus2 = FT_OpenEx("FT999999", FT_OPEN_BY_SERIAL_NUMBER, &ftHandle2);  
if (ftStatus == FT_OK && ftStatus2 == FT_OK) {  
    // success - both devices are open  
}  
else {  
    // failure - one or both of the devices has not been opened  
}
```

4. Open 2 devices with descriptions "USB Serial Converter" and "USB Pump Controller"

```
ftStatus = FT_OpenEx("USB Serial Converter", FT_OPEN_BY_DESCRIPTION, &ftHandle1);  
ftStatus2 = FT_OpenEx("USB Pump Controller", FT_OPEN_BY_DESCRIPTION, &ftHandle2);  
if (ftStatus == FT_OK && ftStatus2 == FT_OK) {  
    // success - both devices are open  
}  
else {  
    // failure - one or both of the devices has not been opened  
}
```

5. Open a device at location 23

```
dwLoc = 0x23;
ftStatus = FT_OpenEx(dwLoc, FT_OPEN_BY_LOCATION, &ftHandle1);
if (ftStatus == FT_OK) {
    // success - device at location 23 is open
}
else {
    // failure
}
```

6. Open 2 devices at locations 23 and 31

```
dwLoc = 0x23;
ftStatus = FT_OpenEx(dwLoc, FT_OPEN_BY_LOCATION, &ftHandle1);
dwLoc = 0x31;
ftStatus2 = FT_OpenEx(dwLoc, FT_OPEN_BY_LOCATION, &ftHandle2);
if (ftStatus == FT_OK && ftStatus2 == FT_OK) {
    // success - both devices are open
}
else {
    // failure - one or both of the devices has not been opened
}
```

3.9 FT_Close

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Close an open device.

Definition

FT_STATUS **FT_Close** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if (ftStatus == FT_OK) {
    // FT_Open OK, use ftHandle to access device
}
```

```
// when finished, call FT_Close
FT_Close(ftHandle);
}
else {
    // FT_Open failed
}
```

3.10 FT_Read

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Read data from the device.

Definition

FT_STATUS **FT_Read** (FT_HANDLE *ftHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToRead*,
LPDWORD *lpdwBytesReturned*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpBuffer</i>	Pointer to the buffer that receives the data from the device.
<i>dwBytesToRead</i>	Number of bytes to be read from the device.
<i>lpdwBytesReturned</i>	Pointer to a variable of type DWORD which receives the number of bytes read from the device.

Return Value

FT_OK if successful, FT_IO_ERROR otherwise.

Remarks

FT_Read always returns the number of bytes read in *lpdwBytesReturned*.

This function does not return until *dwBytesToRead* bytes have been read into the buffer. The number of bytes in the receive queue can be determined by calling [FT_GetStatus](#) or [FT_GetQueueStatus](#), and passed to FT_Read as *dwBytesToRead* so that the function reads the device and returns immediately.

When a read timeout value has been specified in a previous call to [FT_SetTimeouts](#), FT_Read returns when the timer expires or *dwBytesToRead* have been read, whichever occurs first. If the timeout occurred, FT_Read reads available data into the buffer and returns FT_OK.

An application should use the function return value and *lpdwBytesReturned* when processing the buffer. If the return value is FT_OK, and *lpdwBytesReturned* is equal to *dwBytesToRead* then FT_Read has completed normally. If the return value is FT_OK, and *lpdwBytesReturned* is less than *dwBytesToRead* then a timeout has occurred and the read has been partially completed. Note that if a timeout occurred and no data was read, the return value is still FT_OK.

A return value of FT_IO_ERROR suggests an error in the parameters of the function, or a fatal error like a USB disconnect has occurred.

Examples

1. This sample shows how to read all the data currently available.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
DWORD EventDWord;
DWORD TxBytes;
DWORD RxBytes;
DWORD BytesReceived;
char RxBuffer[256];

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}

FT_GetStatus(ftHandle, &RxBytes, &TxBytes, &EventDWord);
if (RxBytes > 0) {
    ftStatus = FT_Read(ftHandle, RxBuffer, RxBytes, &BytesReceived);
    if (ftStatus == FT_OK) {
        // FT_Read OK
    }
    else {
        // FT_Read Failed
    }
}

FT_Close(ftHandle);
```

2. This sample shows how to read with a timeout of 5 seconds.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
DWORD RxBytes = 10;
DWORD BytesReceived;
char RxBuffer[256];

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}

FT_SetTimeouts(ftHandle, 5000, 0);
ftStatus = FT_Read(ftHandle, RxBuffer, RxBytes, &BytesReceived);
if (ftStatus == FT_OK) {
    if (BytesReceived == RxBytes) {
        // FT_Read OK
    }
    else {
        // FT_Read Timeout
    }
}

else {
    // FT_Read Failed
}

FT_Close(ftHandle);
```


3.11 FT_Write

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

Write data to the device.

Definition

FT_STATUS **FT_Write** (FT_HANDLE *ftHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToWrite*, LPDWORD *lpdwBytesWritten*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpBuffer</i>	Pointer to the buffer that contains the data to be written to the device.
<i>dwBytesToWrite</i>	Number of bytes to write to the device.
<i>lpdwBytesWritten</i>	Pointer to a variable of type DWORD which receives the number of bytes written to the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

```
FT_HANDLE ftHandle;  
FT_STATUS ftStatus;  
DWORD BytesWritten;  
char TxBuffer[256]; // Contains data to write to device  
  
ftStatus = FT_Open(0, &ftHandle);  
if(ftStatus != FT_OK) {  
    // FT_Open failed  
    return;  
}  
  
ftStatus = FT_Write(ftHandle, TxBuffer, sizeof(TxBuffer), &BytesWritten);  
if (ftStatus == FT_OK) {  
    // FT_Write OK  
}  
else {  
    // FT_Write Failed  
}  
}  
FT_Close(ftHandle);
```

3.12 FT_SetBaudRate

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function sets the baud rate for the device.

Definition

FT_STATUS **FT_SetBaudRate** (FT_HANDLE *ftHandle*, DWORD *dwBaudRate*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>dwBaudRate</i>	Baud rate.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}

ftStatus = FT_SetBaudRate(ftHandle, 115200); // Set baud rate to 115200
if (ftStatus == FT_OK) {
    // FT_SetBaudRate OK
}
else {
    // FT_SetBaudRate Failed
}

FT_Close(ftHandle);
```

3.13 FT_SetDivisor

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

This function sets the baud rate for the device. It is used to set non-standard baud rates.

Definition

FT_STATUS **FT_SetDivisor** (FT_HANDLE *ftHandle*, USHORT *usDivisor*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>usDivisor</i>	Divisor.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is no longer required as [FT_SetBaudRate](#) will now automatically calculate the required divisor for a requested baud rate. The application note "Setting baud rates for the FT8U232AM" is available from the Application Notes section of the FTDI website describes how to calculate the divisor for a non-standard baud rate.

3.14 FT_SetDataCharacteristics

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

This function sets the data characteristics for the device.

Definition

FT_STATUS **FT_SetDataCharacteristics** (FT_HANDLE *ftHandle*, UCHAR *uWordLength*,
UCHAR *uStopBits*, UCHAR *uParity*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>uWordLength</i>	Number of bits per word - must be FT_BITS_8 or FT_BITS_7 .
<i>uStopBits</i>	Number of stop bits - must be FT_STOP_BITS_1 or FT_STOP_BITS_2 .
<i>uParity</i>	Parity - must be FT_PARITY_NONE , FT_PARITY_ODD , FT_PARITY_EVEN , FT_PARITY_MARK or FT_PARITY_SPACE .

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}

// Set 8 data bits, 1 stop bit and no parity
ftStatus = FT_SetDataCharacteristics(ftHandle, FT_BITS_8, FT_STOP_BITS_1,
FT_PARITY_NONE);
    if (ftStatus == FT_OK) {
        // FT_SetDataCharacteristics OK
    }
    else {
        // FT_SetDataCharacteristics Failed
    }
}

FT_Close(ftHandle);
```

3.15 FT_SetTimeouts

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

This function sets the read and write timeouts for the device.

Definition

FT_STATUS **FT_SetTimeouts** (FT_HANDLE *ftHandle*, DWORD *dwReadTimeout*,
DWORD *dwWriteTimeout*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>dwReadTimeout</i>	Read timeout in milliseconds.
<i>dwWriteTimeout</i>	Write timeout in milliseconds.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}
// Set read timeout of 5sec, write timeout of 1sec
ftStatus = FT_SetTimeouts(ftHandle, 5000, 1000);
if (ftStatus == FT_OK) {
    // FT_SetTimeouts OK
}
else {
    // FT_SetTimeouts failed
}

FT_Close(ftHandle);
```

3.16 FT_SetFlowControl

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

This function sets the flow control for the device.

Definition

FT_STATUS **FT_SetFlowControl** (FT_HANDLE *ftHandle*, USHORT *usFlowControl*,
UCHAR *uXon*, UCHAR *uXoff*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>usFlowControl</i>	Must be one of FT_FLOW_NONE , FT_FLOW_RTS_CTS , FT_FLOW_DTR_DSR or FT_FLOW_XON_XOFF .

uXon Character used to signal Xon. Only used if flow control is [FT_FLOW_XON_XOFF](#).

uXoff Character used to signal Xoff. Only used if flow control is [FT_FLOW_XON_XOFF](#).

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}

// Set RTS/CTS flow control
ftStatus = FT_SetFlowControl(ftHandle, FT_FLOW_RTS_CTS, 0x11, 0x13);
if (ftStatus == FT_OK) {
    // FT_SetFlowControl OK
}
else {
    // FT_SetFlowControl Failed
}

FT_Close(ftHandle);
```

3.17 FT_SetDtr

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function sets the Data Terminal Ready (DTR) control signal.

Definition

FT_STATUS **FT_SetDtr** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function asserts the Data Terminal Ready (DTR) line of the device.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}
ftStatus = FT_SetDtr(ftHandle);
if (ftStatus == FT_OK) {
    // FT_SetDtr OK
}
else {
    // FT_SetDtr failed
}

FT_Close(ftHandle);
```

3.18 FT_ClrDtr

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function clears the Data Terminal Ready (DTR) control signal.

Definition

FT_STATUS **FT_ClrDtr** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function de-asserts the Data Terminal Ready (DTR) line of the device.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
```

```
ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}

ftStatus = FT_ClrDtr(ftHandle);
if (ftStatus == FT_OK) {
    // FT_ClrDtr OK
}
else {
    // FT_ClrDtr failed
}

FT_Close(ftHandle);
```

3.19 FT_SetRts

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function sets the Request To Send (RTS) control signal.

Definition

FT_STATUS **FT_SetRts** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function asserts the Request To Send (RTS) line of the device.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}
```



```
ftStatus = FT_SetRts(ftHandle);  
if (ftStatus == FT_OK) {  
    // FT_SetRts OK  
}  
else {  
    // FT_SetRts failed  
}  
  
FT_Close(ftHandle);
```

3.20 FT_ClrRts

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function clears the Request To Send (RTS) control signal.

Definition

FT_STATUS **FT_ClrRts** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function de-asserts the Request To Send (RTS) line of the device.

Example

```
FT_HANDLE ftHandle;  
FT_STATUS ftStatus;  
  
ftStatus = FT_Open(0, &ftHandle);  
if(ftStatus != FT_OK) {  
    // FT_Open failed  
    return;  
}  
  
ftStatus = FT_ClrRts(ftHandle);  
if (ftStatus == FT_OK) {  
    // FT_ClrRts OK  
}  
else {  
    // FT_ClrRts failed
```

```
}  
  
FT_Close(ftHandle);
```

3.21 FT_GetModemStatus

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

Gets the modem status and line status from the device.

Definition

FT_STATUS **FT_GetModemStatus** (FT_HANDLE *ftHandle*, LPDWORD *lpdwModemStatus*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpdwModemStatus</i>	Pointer to a variable of type DWORD which receives the modem status and line status from the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

The least significant byte of the *lpdwModemStatus* value holds the modem status. On Windows and Windows CE, the line status is held in the second least significant byte of the *lpdwModemStatus* value.

The modem status is bit-mapped as follows: Clear To Send ([CTS](#)) = 0x10, Data Set Ready ([DSR](#)) = 0x20, Ring Indicator ([RI](#)) = 0x40, Data Carrier Detect ([DCD](#)) = 0x80.

The line status is bit-mapped as follows: Overrun Error ([OE](#)) = 0x02, Parity Error ([PE](#)) = 0x04, Framing Error ([FE](#)) = 0x08, Break Interrupt ([BI](#)) = 0x10.

Example

```
FT_HANDLE ftHandle;  
FT_STATUS ftStatus;  
DWORD dwModemStatus = 0;  
DWORD dwLineStatus = 0;  
  
ftStatus = FT_Open(0, &ftHandle);  
if(ftStatus != FT_OK) {  
    // FT_Open failed  
    return;  
}
```

```
ftStatus = FT_GetModemStatus(ftHandle, &dwModemStatus);
if (ftStatus == FT_OK) {
    // FT_GetModemStatus OK
    // Line status is the second byte of the dwModemStatus value
    dwLineStatus = ((dwModemStatus >> 8) & 0x000000FF);
    // Now mask off the modem status byte
    dwModemStatus = (dwModemStatus & 0x000000FF);
}
else {
    // FT_GetModemStatus failed
}

FT_Close(ftHandle);
```

3.22 FT_GetQueueStatus

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Gets the number of bytes in the receive queue.

Definition

FT_STATUS **FT_GetQueueStatus** (FT_HANDLE *ftHandle*, LPDWORD *lpdwAmountInRxQueue*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpdwAmountInRxQueue</i>	Pointer to a variable of type DWORD which receives the number of bytes in the receive queue.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
DWORD RxBytes;
DWORD BytesReceived;
char RxBuffer[256];

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}

FT_GetQueueStatus(ftHandle, &RxBytes);
if (RxBytes > 0) {
    ftStatus = FT_Read(ftHandle, RxBuffer, RxBytes, &BytesReceived);
```

```
    if (ftStatus == FT_OK) {  
        // FT_Read OK  
    }  
    else {  
        // FT_Read Failed  
    }  
}  
  
FT_Close(ftHandle);
```

3.23 FT_GetDeviceInfo

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Get device information for an open device.

Definition

FT_STATUS **FT_GetDeviceInfo** (FT_HANDLE *ftHandle*, FT_DEVICE **pftType*,
LPDWORD *lpdwID*, PCHAR *pcSerialNumber*,
PCHAR *pcDescription*, PVOID *pvDummy*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>pftType</i>	Pointer to unsigned long to store device type.
<i>lpdwID</i>	Pointer to unsigned long to store device ID.
<i>pcSerialNumber</i>	Pointer to buffer to store device serial number as a null-terminated string.
<i>pcDescription</i> string.	Pointer to buffer to store device description as a null-terminated string.
<i>pvDummy</i>	Reserved for future use - should be set to NULL.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to return the device type, device ID, device description and serial number.

The device ID is encoded in a DWORD - the most significant word contains the vendor ID, and the least significant word contains the product ID. So the returned ID 0x04036001 corresponds to the device ID VID_0403&PID_6001.

Example

```
FT_HANDLE ftHandle;
FT_DEVICE ftDevice;
FT_STATUS ftStatus;
DWORD deviceID;
char SerialNumber[16];
char Description[64];

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}

ftStatus = FT_GetDeviceInfo(
    ftHandle,
    &ftDevice,
    &deviceID,
    SerialNumber,
    Description,
    NULL
);

if (ftStatus == FT_OK) {
    if (ftDevice == FT_DEVICE_232H)
        ; // device is FT232H
    else if (ftDevice == FT_DEVICE_4232H)
        ; // device is FT4232H
    else if (ftDevice == FT_DEVICE_2232H)
        ; // device is FT2232H
    else if (ftDevice == FT_DEVICE_232R)
        ; // device is FT232R
    else if (ftDevice == FT_DEVICE_2232C)
        ; // device is FT2232C/L/D
    else if (ftDevice == FT_DEVICE_BM)
        ; // device is FTU232BM
    else if (ftDevice == FT_DEVICE_AM)
        ; // device is FT8U232AM
    else
        ; // unknown device (this should not happen!)
    // deviceID contains encoded device ID
    // SerialNumber, Description contain 0-terminated strings
}
else {
    // FT_GetDeviceType FAILED!
}

FT_Close(ftHandle);
```

3.24 FT_GetDriverVersion

Supported Operating Systems

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function returns the D2XX driver version number.

Definition

FT_STATUS **FT_GetDriverVersion** (FT_HANDLE *ftHandle*, LPDWORD *lpdwDriverVersion*)

Parameters

ftHandle Handle of the device.
lpdwDriverVersion Pointer to the driver version number.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

A version number consists of major, minor and build version numbers contained in a 4-byte field (unsigned long). Byte0 (least significant) holds the build version, Byte1 holds the minor version, and Byte2 holds the major version. Byte3 is currently set to zero.

For example, driver version "2.04.06" is represented as 0x00020406. Note that a device has to be opened before this function can be called.

Example

```
FT_HANDLE ftHandle;  
FT_STATUS ftStatus;  
DWORD dwDriverVer;  
  
// Get driver version  
ftStatus = FT_Open(0,&ftHandle);  
if (ftStatus == FT_OK) {  
    ftStatus = FT_GetDriverVersion(ftHandle,&dwDriverVer);  
    if (ftStatus == FT_OK)  
        printf("Driver version = 0x%x\n",dwDriverVer);  
    else  
        printf("error reading driver version\n");  
    FT_Close(ftHandle);  
}
```

3.25 FT_GetLibraryVersion

Supported Operating Systems

Windows (2000 and later)
Windows CE (4.2 and later)

Summary

This function returns D2XX DLL version number.

Definition

FT_STATUS **FT_GetLibraryVersion** (LPDWORD *lpdwDLLVersion*)

Parameters

lpdwDLLVersion Pointer to the DLL version number.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

A version number consists of major, minor and build version numbers contained in a 4-byte field (unsigned long). Byte0 (least significant) holds the build version, Byte1 holds the minor version, and Byte2 holds the major version. Byte3 is currently set to zero.

For example, D2XX DLL version "3.01.15" is represented as 0x00030115. Note that this function does not take a handle, and so it can be called without opening a device.

Example

```
FT_STATUS ftStatus;
DWORD dwLibraryVer;

// Get DLL version
ftStatus = FT_GetLibraryVersion(&dwLibraryVer);
if (ftStatus == FT_OK)
    printf("Library version = 0x%x\n", dwLibraryVer);
else
    printf("error reading library version\n");
```

3.26 FT_GetComPortNumber

Supported Operating Systems

Windows (2000 and later)

Summary

Retrieves the COM port associated with a device.

Definition

FT_STATUS **FT_GetComPortNumber** (FT_HANDLE *ftHandle*, LPLONG *lplComPortNumber*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lplComPortNumber</i>	Pointer to a variable of type LONG which receives the COM port number associated with the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is only available when using the Windows CDM driver as both the D2XX and VCP drivers can be installed at the same time.

If no COM port is associated with the device, *lplComPortNumber* will have a value of -1.

Example

```
FT_HANDLE ftHandle; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;
LONG lComPortNumber;
```

```
ftStatus = FT_GetComPortNumber(ftHandle, &lComPortNumber);  
if (status == FT_OK) {  
    if (lComPortNumber == -1) {  
        // No COM port assigned  
    }  
    else {  
        // COM port assigned with number held in lComPortNumber  
    }  
}  
else {  
    // FT_GetComPortNumber FAILED!  
}  
FT_Close(ftHandle);
```

3.27 FT_GetStatus

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Gets the device status including number of characters in the receive queue, number of characters in the transmit queue, and the current event status.

Definition

FT_STATUS **FT_GetStatus** (FT_HANDLE *ftHandle*, LPDWORD *lpdwAmountInRxQueue*,
LPDWORD *lpdwAmountInTxQueue*, LPDWORD *lpdwEventStatus*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpdwAmountInRxQueue</i>	Pointer to a variable of type DWORD which receives the number of characters in the receive queue.
<i>lpdwAmountInTxQueue</i>	Pointer to a variable of type DWORD which receives the number of characters in the transmit queue.
<i>lpdwEventStatus</i> the event status.	Pointer to a variable of type DWORD which receives the current state of

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

For an example of how to use this function, see the sample code in [FT_SetEventNotification](#).

3.28 FT_SetEventNotification

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

Sets conditions for event notification.

Definition

FT_STATUS **FT_SetEventNotification** (FT_HANDLE *ftHandle*, DWORD *dwEventMask*, PVOID *pvArg*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>dwEventMask</i>	Conditions that cause the event to be set.
<i>pvArg</i>	Interpreted as the handle of an event.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

An application can use this function to setup conditions which allow a thread to block until one of the conditions is met. Typically, an application will create an event, call this function, then block on the event. When the conditions are met, the event is set, and the application thread unblocked.

dwEventMask is a bit-map that describes the events the application is interested in. *pvArg* is interpreted as the handle of an event which has been created by the application. If one of the event conditions is met, the event is set.

If [FT_EVENT_RXCHAR](#) is set in *dwEventMask*, the event will be set when a character has been received by the device.

If [FT_EVENT_MODEM_STATUS](#) is set in *dwEventMask*, the event will be set when a change in the modem signals has been detected by the device.

If [FT_EVENT_LINE_STATUS](#) is set in *dwEventMask*, the event will be set when a change in the line status has been detected by the device.

Examples

1. This example is valid for Windows and Windows CE and shows how to wait for a character to be received or a change in modem status.

```
// First, create the event and call FT_SetEventNotification.
FT_HANDLE ftHandle; // handle of an open device
FT_STATUS ftStatus;
HANDLE hEvent;
DWORD EventMask;

hEvent = CreateEvent(
    NULL,
    false, // auto-reset event
    false, // non-signalled state
```

```
        ""
    );
EventMask = FT_EVENT_RXCHAR | FT_EVENT_MODEM_STATUS;
ftStatus = FT_SetEventNotification(ftHandle, EventMask, hEvent);

// Sometime later, block the application thread by waiting on the event, then when the
// event has
// occurred, determine the condition that caused the event, and process it accordingly.
WaitForSingleObject(hEvent, INFINITE);

DWORD EventDWord;
DWORD RxBytes;
DWORD TxBytes;

FT_GetStatus(ftHandle, &RxBytes, &TxBytes, &EventDWord);
if (EventDWord & FT_EVENT_MODEM_STATUS) {
    // modem status event detected, so get current modem status
    FT_GetModemStatus(ftHandle, &Status);
    if (Status & 0x00000010) {
        // CTS is high
    }
    else {
        // CTS is low
    }
    if (Status & 0x00000020) {
        // DSR is high
    }
    else {
        // DSR is low
    }
}
if (RxBytes > 0) {
    // call FT_Read() to get received data from device
}
```

2. This example is valid for Linux and shows how to wait for a character to be received or a change in modem status.

```
// First, create the event and call FT_SetEventNotification.
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
EVENT_HANDLE eh;
DWORD EventMask;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}

pthread_mutex_init(&eh.eMutex, NULL);
pthread_cond_init(&eh.eCondVar, NULL);

EventMask = FT_EVENT_RXCHAR | FT_EVENT_MODEM_STATUS;
ftStatus = FT_SetEventNotification(ftHandle, EventMask, (PVOID)&eh);

// Sometime later, block the application thread by waiting on the event, then when the
// event has
// occurred, determine the condition that caused the event, and process it accordingly.

pthread_mutex_lock(&eh.eMutex);
pthread_cond_wait(&eh.eCondVar, &eh.eMutex);
pthread_mutex_unlock(&eh.eMutex);

DWORD EventDWord;
DWORD RxBytes;
```

```
DWORD TxBytes;  
DWORD Status;  
FT_GetStatus(ftHandle, &RxBytes, &TxBytes, &EventDWord);  
if (EventDWord & FT_EVENT_MODEM_STATUS) {  
    // modem status event detected, so get current modem status  
    FT_GetModemStatus(ftHandle, &Status);  
    if (Status & 0x00000010) {  
        // CTS is high  
    }  
    else {  
        // CTS is low  
    }  
    if (Status & 0x00000020) {  
        // DSR is high  
    }  
    else {  
        // DSR is low  
    }  
}  
if (RxBytes > 0) {  
    // call FT_Read() to get received data from device  
}  
  
FT_Close(ftHandle);
```

3.29 FT_SetChars

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function sets the special characters for the device.

Definition

FT_STATUS **FT_SetChars** (FT_HANDLE *ftHandle*, UCHAR *uEventCh*, UCHAR *uEventChEn*, UCHAR *uErrorCh*, UCHAR *uErrorChEn*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>uEventCh</i>	Event character.
<i>uEventChEn</i>	0 if event character disabled, non-zero otherwise.
<i>uErrorCh</i>	Error character.
<i>uErrorChEn</i>	0 if error character disabled, non-zero otherwise.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function allows for inserting specified characters in the data stream to represent events firing or errors occurring.

3.30 FT_SetBreakOn

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Sets the BREAK condition for the device.

Definition

FT_STATUS **FT_SetBreakOn** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

```
FT_HANDLE ftHandle; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_SetBreakOn(ftHandle);
if (ftStatus == FT_OK) {
    // FT_SetBreakOn OK
}
else {
    // FT_SetBreakOn failed
}

FT_Close(ftHandle);
```

3.31 FT_SetBreakOff

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Resets the BREAK condition for the device.

Definition

FT_STATUS **FT_SetBreakOff** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

```
FT_HANDLE ftHandle; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_SetBreakOff(ftHandle);
if (ftStatus == FT_OK) {
    // FT_SetBreakOff OK
}
else {
    // FT_SetBreakOff failed
}

FT_Close(ftHandle);
```

3.32 FT_Purge

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function purges receive and transmit buffers in the device.

Definition

FT_STATUS **FT_Purge** (FT_HANDLE *ftHandle*, DWORD *dwMask*)

Parameters

ftHandle Handle of the device.

uEventCh Combination of [FT_PURGE_RX](#) and [FT_PURGE_TX](#).

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

```
FT_HANDLE ftHandle; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_Purge(ftHandle, FT_PURGE_RX | FT_PURGE_TX); // Purge both Rx and Tx buffers
if (ftStatus == FT_OK) {
    // FT_Purge OK
}
else {
    // FT_Purge failed
}
FT_Close(ftHandle);
```

3.33 FT_ResetDevice

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function sends a reset command to the device.

Definition

FT_STATUS **FT_ResetDevice** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Example

```
FT_HANDLE ftHandle; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_ResetDevice(ftHandle);
if (ftStatus == FT_OK) {
    // FT_ResetDevice OK
}
else {
    // FT_ResetDevice failed
}
FT_Close(ftHandle);
```

3.34 FT_ResetPort

Supported Operating Systems

Windows (2000 and later)

Summary

Send a reset command to the port.

Definition

FT_STATUS **FT_ResetPort** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to attempt to recover the port after a failure. It is not equivalent to an unplug-replug event. For the equivalent of an unplug-replug event, use [FT_CyclePort](#).

Example

```
FT_HANDLE ftHandle; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_ResetPort(ftHandle);
if (ftStatus == FT_OK) {
    // Port has been reset
}
else {
    // FT_ResetPort FAILED!
}
```

3.35 FT_CyclePort

Supported Operating Systems

Windows (2000 and later)

Summary

Send a cycle command to the USB port.

Definition

FT_STATUS **FT_CyclePort** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

The effect of this function is the same as disconnecting then reconnecting the device from USB. Possible use of this function is situations where a fatal error has occurred and it is difficult, or not possible, to recover without unplugging and replugging the USB cable. This function can also be used after re-programming the EEPROM to force the FTDI device to read the new EEPROM contents which would otherwise require a physical disconnect-reconnect.

As the current session is not restored when the driver is reloaded, the application must be able to recover after calling this function. It is the responsibility of the application to close the handle after successfully calling FT_CyclePort.

For FT4232H, FT2232H and FT2232 devices, FT_CyclePort will only work under Windows XP and later.

Example

```
FT_HANDLE ftHandle; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_CyclePort(ftHandle);
if (ftStatus == FT_OK) {
    // Port has been cycled.
    // Close the handle.
    ftStatus = FT_Close(ftHandle);
}
else {
    // FT_CyclePort FAILED!
}
```

3.36 FT_Rescan

Supported Operating Systems

Windows (2000 and later)

Summary

This function can be of use when trying to recover devices programmatically.

Definition

FT_STATUS **FT_Rescan** ()

Parameters

None

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

Calling FT_Rescan is equivalent to clicking the "Scan for hardware changes" button in the Device Manager. Only USB hardware is checked for new devices. All USB devices are scanned, not just FTDI devices.

Example

```
FT_STATUS ftstatus;

ftStatus = FT_Rescan();
if(ftStatus != FT_OK) {
    // FT_Rescan failed!
    return;
}
```

3.37 FT_Reload

Supported Operating Systems

Windows (2000 and later)

Summary

This function forces a reload of the driver for devices with a specific VID and PID combination.

Definition

FT_STATUS **FT_Reload** (WORD *wVID*, WORD *wPID*)

Parameters

<i>wVID</i>	Vendor ID of the devices to reload the driver for.
<i>wPID</i>	Product ID of the devices to reload the driver for.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

Calling FT_Reload forces the operating system to unload and reload the driver for the specified device IDs. If the VID and PID parameters are null, the drivers for USB root hubs will be reloaded, causing all USB devices connected to reload their drivers. Please note that this function will not work correctly on 64-bit Windows when called from a 32-bit application.

Examples

1. This example shows how to call FT_Reload to reload the driver for a standard FT232R device (VID 0x0403, PID 0x6001).

```
FT_STATUS ftstatus;
WORD wVID = 0x0403;
WORD wPID = 0x6001;
```

```
ftStatus = FT_Reload(wVID,wPID);  
if(ftStatus != FT_OK) {  
    // FT_Reload failed!  
    return;  
}
```

2. This example shows how to call FT_Reload to reload the drivers for all USB devices.

```
FT_STATUS ftstatus;  
WORD wVID = 0x0000;  
WORD wPID = 0x0000;  
  
ftStatus = FT_Reload(wVID,wPID);  
if(ftStatus != FT_OK) {  
    // FT_Reload failed!  
    return;  
}
```

3.38 FT_SetResetPipeRetryCount

Supported Operating Systems

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Set the ResetPipeRetryCount value.

Definition

FT_STATUS **FT_SetResetPipeRetryCount** (FT_HANDLE *ftHandle*, DWORD *dwCount*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>dwCount</i>	Unsigned long containing required ResetPipeRetryCount.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to set the ResetPipeRetryCount. ResetPipeRetryCount controls the maximum number of times that the driver tries to reset a pipe on which an error has occurred. ResetPipeRequestRetryCount defaults to 50. It may be necessary to increase this value in noisy environments where a lot of USB errors occur.

Example

```
FT_HANDLE ftHandle; // valid handle returned from FT_OpenEx  
FT_STATUS ftStatus;
```

```
DWORD dwRetryCount;  
  
dwRetryCount = 100;  
ftStatus = FT_SetResetPipeRetryCount(ftHandle, dwRetryCount);  
if (ftStatus == FT_OK) {  
    // ResetPipeRetryCount set to 100  
}  
else {  
    // FT_SetResetPipeRetryCount FAILED!  
}
```

3.39 FT_StopInTask

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Stops the driver's IN task.

Definition

FT_STATUS **FT_StopInTask** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to put the driver's IN task (read) into a wait state. It can be used in situations where data is being received continuously, so that the device can be purged without more data being received. It is used together with [FT_RestartInTask](#) which sets the IN task running again.

Example

```
FT_HANDLE ftHandle;  
FT_STATUS ftStatus;  
  
ftStatus = FT_Open(0, &ftHandle);  
if(ftStatus != FT_OK) {  
    // FT_Open failed  
    return;  
}  
  
do {  
    ftStatus = FT_StopInTask(ftHandle);  
} while (ftStatus != FT_OK);
```

```
//  
// Do something - for example purge device  
//  
  
do {  
    ftStatus = FT_RestartInTask(ftHandle);  
} while (ftStatus != FT_OK);  
  
FT_Close(ftHandle);
```

3.40 FT_RestartInTask

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Restart the driver's IN task.

Definition

FT_STATUS **FT_RestartInTask** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to restart the driver's IN task (read) after it has been stopped by a call to [FT_StopInTask](#).

Example

```
FT_HANDLE ftHandle;  
FT_STATUS ftStatus;  
  
ftStatus = FT_Open(0, &ftHandle);  
if(ftStatus != FT_OK) {  
    // FT_Open failed  
    return;  
}  
  
do {  
    ftStatus = FT_StopInTask(ftHandle);  
} while (ftStatus != FT_OK);  
  
//
```

```
// Do something - for example purge device
//

do {
    ftStatus = FT_RestartInTask(ftHandle);
} while (ftStatus != FT_OK);

FT_Close(ftHandle);
```

3.41 FT_SetDeadmanTimeout

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function allows the maximum time in milliseconds that a USB request can remain outstanding to be set.

Definition

FT_STATUS **FT_SetDeadmanTimeout** (FT_HANDLE *ftHandle*, DWORD *dwDeadmanTimeout*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>dwDeadmanTimeout</i>	Deadman timeout value in milliseconds. Default value is 5000.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

The deadman timeout is referred to in application note AN232B-10 Advanced Driver Options from the FTDI web site as the USB timeout. It is unlikely that this function will be required by most users.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
DWORD dwDeadmanTimeout = 6000;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}

ftStatus = FT_SetDeadmanTimeout(ftHandle, dwDeadmanTimeout);
if (ftStatus == FT_OK) {
```

```
        // Set Deadman Timer to 6 seconds
    }
    else {
        // FT_SetDeadmanTimeout FAILED!
    }

    FT_Close(ftHandle);
```

3.42 FT_IoCtl

Undocumented function.

3.43 FT_SetWaitMask

Undocumented function.

3.44 FT_WaitOnMask

Undocumented function.

4 EEPROM Programming Interface Functions

FTDI device EEPROMs can be both read and programmed using the functions listed in this section.

4.1 FT_ReadEE

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

Read a value from an EEPROM location.

Definition

FT_STATUS **FT_ReadEE** (FT_HANDLE *ftHandle*, DWORD *dwWordOffset*, LPWORD *lpwValue*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>dwWordOffset</i>	EEPROM location to read from.
<i>lpwValue</i>	Pointer to the WORD value read from the EEPROM.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

EEPROMs for FTDI devices are organised by WORD, so each value returned is 16-bits wide.

4.2 FT_WriteEE

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

Write a value to an EEPROM location.

Definition

FT_STATUS **FT_WriteEE** (FT_HANDLE *ftHandle*, DWORD *dwWordOffset*, WORD *wValue*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>dwWordOffset</i>	EEPROM location to read from.
<i>wValue</i>	The WORD value write to the EEPROM.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

EEPROMs for FTDI devices are organised by WORD, so each value written to the EEPROM is 16-bits wide.

4.3 FT_EraseEE

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

Erases the device EEPROM.

Definition

FT_STATUS **FT_EraseEE** (FT_HANDLE *ftHandle*)

Parameters

<i>ftHandle</i>	Handle of the device.
-----------------	-----------------------

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function will erase the entire contents of an EEPROM, including the user area. Note that the FT232R and FT245R devices have an internal EEPROM that cannot be erased.

4.4 FT_EE_Read

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

Read the contents of the EEPROM.

Definition

FT_STATUS **FT_EE_Read** (FT_HANDLE *ftHandle*, PFT_PROGRAM_DATA *pData*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>pData</i>	Pointer to structure of type FT_PROGRAM_DATA.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function interprets the parameter *pData* as a pointer to a structure of type *FT_PROGRAM_DATA* that contains storage for the data to be read from the EEPROM.

The function does not perform any checks on buffer sizes, so the buffers passed in the *FT_PROGRAM_DATA* structure must be big enough to accommodate their respective strings (including null terminators). The sizes shown in the following example are more than adequate and can be rounded down if necessary. The restriction is that the *Manufacturer* string length plus the *Description* string length is less than or equal to 40 characters.

Note that the DLL must be informed which version of the *FT_PROGRAM_DATA* structure is being used. This is done through the *Signature1*, *Signature2* and *Version* elements of the structure. *Signature1* should always be *0x00000000*, *Signature2* should always be *0xFFFFFFFF* and *Version* can be set to use whichever version is required. For compatibility with all current devices *Version* should be set to the latest version of the *FT_PROGRAM_DATA* structure which is defined in FTD2XX.h.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus = FT_Open(0, &ftHandle);
if (ftStatus != FT_OK) {
    // FT_Open FAILED!
}

FT_PROGRAM_DATA ftData;
char ManufacturerBuf[32];
char ManufacturerIdBuf[16];
char DescriptionBuf[64];
char SerialNumberBuf[16];

ftData.Signature1 = 0x00000000;
ftData.Signature2 = 0xffffffff;
ftData.Version = 0x00000005;           // EEPROM structure with FT232H extensions
ftData.Manufacturer = ManufacturerBuf;
ftData.ManufacturerId = ManufacturerIdBuf;
```

```
ftData.Description = DescriptionBuf;
ftData.SerialNumber = SerialNumberBuf;

ftStatus = FT_EE_Read(ftHandle, &ftData);
if (ftStatus == FT_OK) {
    // FT_EE_Read OK, data is available in ftData
}
else {
    // FT_EE_Read FAILED!
}
FT_Close(ftHandle);
```

4.5 FT_EE_ReadEx

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Read the contents of the EEPROM and pass strings separately.

Definition

FT_STATUS **FT_EE_ReadEx** (FT_HANDLE *ftHandle*, PFT_PROGRAM_DATA *pData*,
char **Manufacturer*, char **ManufacturerId*, char **Description*,
char **SerialNumber*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>pData</i>	Pointer to structure of type FT_PROGRAM_DATA.
* <i>Manufacturer</i>	Pointer to a null-terminated string containing the manufacturer name.
* <i>ManufacturerId</i>	Pointer to a null-terminated string containing the manufacturer ID.
* <i>Description</i>	Pointer to a null-terminated string containing the device description.
* <i>SerialNumber</i>	Pointer to a null-terminated string containing the device serial number.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This variation of the standard [FT_EE_Read](#) function was included to provide support for languages such as LabVIEW where problems can occur when string pointers are contained in a structure.

This function interprets the parameter *pData* as a pointer to a structure of type *FT_PROGRAM_DATA* that contains storage for the data to be read from the EEPROM.

The function does not perform any checks on buffer sizes, so the buffers passed in the *FT_PROGRAM_DATA* structure must be big enough to accommodate their respective strings (including null terminators).

Note that the DLL must be informed which version of the *FT_PROGRAM_DATA* structure is being used. This is done through the *Signature1*, *Signature2* and *Version* elements of the structure. *Signature1* should always be *0x00000000*, *Signature2* should always be *0xFFFFFFFF* and *Version* can be set to use whichever version is required. For compatibility with all current devices *Version* should be set to the latest version of the *FT_PROGRAM_DATA* structure which is defined in FTD2XX.h.

The string parameters in the *FT_PROGRAM_DATA* structure should be passed as DWORDs to avoid overlapping of parameters. All string pointers are passed out separately from the *FT_PROGRAM_DATA* structure.

4.6 FT_EE_Program

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Program the EEPROM.

Definition

FT_STATUS **FT_EE_Program** (FT_HANDLE *ftHandle*, PFT_PROGRAM_DATA *pData*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>pData</i>	Pointer to structure of type FT_PROGRAM_DATA.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function interprets the parameter *pData* as a pointer to a structure of type *FT_PROGRAM_DATA* that contains the data to write to the EEPROM. The data is written to EEPROM, then read back and verified.

If the *SerialNumber* field in *FT_PROGRAM_DATA* is NULL, or *SerialNumber* points to a NULL string, a serial number based on the *ManufacturerId* and the current date and time will be generated. The *Manufacturer* string length plus the *Description* string length must be less than or equal to 40 characters.

Note that the DLL must be informed which version of the *FT_PROGRAM_DATA* structure is being used. This is done through the *Signature1*, *Signature2* and *Version* elements of the structure. *Signature1* should always be *0x00000000*, *Signature2* should always be *0xFFFFFFFF* and *Version* can be set to use whichever version is required. For compatibility with all current devices *Version* should be set to the latest version of the *FT_PROGRAM_DATA* structure which is defined in FTD2XX.h.

If *pData* is NULL, the structure version will default to 0 (original BM series) and the device will be programmed with the default data:

Example

This example shows how to program the EEPROM of an FT232B device. Other parameters would need to be set up for other device types.

```
// Version 4 structure for programming a BM device.  
// Other elements would need non-zero values for FT2232, FT232R, FT245R, FT2232H or  
// FT4232H devices.
```

```
FT_PROGRAM_DATA ftData = {  
    0x00000000,           // Header - must be 0x00000000  
    0xFFFFFFFF,           // Header - must be 0xffffffff  
    0x00000004,           // Header - FT_PROGRAM_DATA version  
    0x0403,               // VID  
    0x6001,               // PID  
    "FTDI",               // Manufacturer  
    "FT",                 // Manufacturer ID  
    "USB HS Serial Converter", // Description  
    "FT000001",           // Serial Number  
    44,                   // MaxPower  
    1,                    // PnP  
    0,                    // SelfPowered  
    1,                    // RemoteWakeup  
    1,                    // non-zero if Rev4 chip, zero otherwise  
    0,                    // non-zero if in endpoint is isochronous  
    0,                    // non-zero if out endpoint is isochronous  
    0,                    // non-zero if pull down enabled  
    1,                    // non-zero if serial number to be used  
    0,                    // non-zero if chip uses USBVersion  
    0x0110                // BCD (0x0200 => USB2)  
    //  
    // FT2232C extensions (Enabled if Version = 1 or greater)  
    //  
    0,                    // non-zero if Rev5 chip, zero otherwise  
    0,                    // non-zero if in endpoint is isochronous  
    0,                    // non-zero if in endpoint is isochronous  
    0,                    // non-zero if out endpoint is isochronous  
    0,                    // non-zero if out endpoint is isochronous  
    0,                    // non-zero if pull down enabled  
    0,                    // non-zero if serial number to be used  
    0,                    // non-zero if chip uses USBVersion  
    0x0,                 // BCD (0x0200 => USB2)  
    0,                    // non-zero if interface is high current  
    0,                    // non-zero if interface is high current  
    0,                    // non-zero if interface is 245 FIFO  
    0,                    // non-zero if interface is 245 FIFO CPU target  
    0,                    // non-zero if interface is Fast serial  
    0,                    // non-zero if interface is to use VCP drivers  
    0,                    // non-zero if interface is 245 FIFO  
    0,                    // non-zero if interface is 245 FIFO CPU target  
    0,                    // non-zero if interface is Fast serial  
    0,                    // non-zero if interface is to use VCP drivers  
    //  
    // FT232R extensions (Enabled if Version = 2 or greater)  
    //  
    0,                    // Use External Oscillator  
    0,                    // High Drive I/Os  
    0,                    // Endpoint size  
    0,                    // non-zero if pull down enabled  
    0,                    // non-zero if serial number to be used  
    0,                    // non-zero if invert TXD  
    0,                    // non-zero if invert RXD  
    0,                    // non-zero if invert RTS  
    0,                    // non-zero if invert CTS  
    0,                    // non-zero if invert DTR  
    0,                    // non-zero if invert DSR  
}
```

```
0, // non-zero if invert DCD
0, // non-zero if invert RI
0, // Cbus Mux control
0, // Cbus Mux control
0, // Cbus Mux control
0, // Cbus Mux control
0, // Cbus Mux control
0, // non-zero if using D2XX drivers
//
// Rev 7 (FT2232H) Extensions (Enabled if Version = 3 or greater)
//
0, // non-zero if pull down enabled
0, // non-zero if serial number to be used
0, // non-zero if AL pins have slow slew
0, // non-zero if AL pins are Schmitt input
0, // valid values are 4mA, 8mA, 12mA, 16mA
0, // non-zero if AH pins have slow slew
0, // non-zero if AH pins are Schmitt input
0, // valid values are 4mA, 8mA, 12mA, 16mA
0, // non-zero if BL pins have slow slew
0, // non-zero if BL pins are Schmitt input
0, // valid values are 4mA, 8mA, 12mA, 16mA
0, // non-zero if BH pins have slow slew
0, // non-zero if BH pins are Schmitt input
0, // valid values are 4mA, 8mA, 12mA, 16mA
0, // non-zero if interface is 245 FIFO
0, // non-zero if interface is 245 FIFO CPU target
0, // non-zero if interface is Fast serial
0, // non-zero if interface is to use VCP drivers
0, // non-zero if interface is 245 FIFO
0, // non-zero if interface is 245 FIFO CPU target
0, // non-zero if interface is Fast serial
0, // non-zero if interface is to use VCP drivers
0, // non-zero if using BCBUS7 to save power for self-
    // powered designs
//
// Rev 8 (FT4232H) Extensions (Enabled if Version = 4)
//
0, // non-zero if pull down enabled
0, // non-zero if serial number to be used
0, // non-zero if AL pins have slow slew
0, // non-zero if AL pins are Schmitt input
0, // valid values are 4mA, 8mA, 12mA, 16mA
0, // non-zero if AH pins have slow slew
0, // non-zero if AH pins are Schmitt input
0, // valid values are 4mA, 8mA, 12mA, 16mA
0, // non-zero if BL pins have slow slew
0, // non-zero if BL pins are Schmitt input
0, // valid values are 4mA, 8mA, 12mA, 16mA
0, // non-zero if BH pins have slow slew
0, // non-zero if BH pins are Schmitt input
0, // valid values are 4mA, 8mA, 12mA, 16mA
0, // non-zero if port A uses RI as RS485 TXDEN
0, // non-zero if port B uses RI as RS485 TXDEN
0, // non-zero if port C uses RI as RS485 TXDEN
0, // non-zero if port D uses RI as RS485 TXDEN
0, // non-zero if interface is to use VCP drivers
0, // non-zero if interface is to use VCP drivers
0, // non-zero if interface is to use VCP drivers
0, // non-zero if interface is to use VCP drivers
//
// Rev 9 (FT232H) Extensions (Enabled if Version = 5)
//
0, // non-zero if pull down enabled
0, // non-zero if serial number to be used
0, // non-zero if AC pins have slow slew
0, // non-zero if AC pins are Schmitt input
```

```
0, // valid values are 4mA, 8mA, 12mA, 16mA
0, // non-zero if AD pins have slow slew
0, // non-zero if AD pins are Schmitt input
0, // valid values are 4mA, 8mA, 12mA, 16mA
0, // Cbus Mux control
0, // Cbus Mux control
0, // Cbus Mux control
0, // Cbus Mux control
0, // Cbus Mux control
0, // Cbus Mux control
0, // Cbus Mux control
0, // Cbus Mux control
0, // Cbus Mux control
0, // non-zero if interface is 245 FIFO
0, // non-zero if interface is 245 FIFO CPU target
0, // non-zero if interface is Fast serial
0, // non-zero if interface is FT1248
0, // FT1248 clock polarity - clock idle high (1) or
    // clock idle low (0)
0, // FT1248 data is LSB (1) or MSB (0)
0, // FT1248 flow control enable
0, // non-zero if interface is to use VCP drivers
0, // non-zero if using ACBUS7 to save power for
    // self-powered designs

};
FT_HANDLE ftHandle;

FT_STATUS ftStatus = FT_Open(0, &ftHandle);
if (ftStatus == FT_OK) {
    ftStatus = FT_EE_Program(ftHandle, &ftData);
    if (ftStatus == FT_OK) {
        // FT_EE_Program OK!
    }
    else {
        // FT_EE_Program FAILED!
    }
    FT_Close(ftHandle);
}
```

4.7 FT_EE_ProgramEx

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Program the EEPROM and pass strings separately.

Definition

FT_STATUS **FT_EE_ProgramEx** (FT_HANDLE *ftHandle*, PFT_PROGRAM_DATA *pData*,
char *Manufacturer, char *ManufacturerId,
char *Description, char *SerialNumber)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>pData</i>	Pointer to structure of type FT_PROGRAM_DATA.
<i>*Manufacturer</i>	Pointer to a null-terminated string containing the manufacturer name.
<i>*ManufacturerId</i>	Pointer to a null-terminated string containing the manufacturer ID.
<i>*Description</i>	Pointer to a null-terminated string containing the device description.
<i>*SerialNumber</i>	Pointer to a null-terminated string containing the device serial number.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This variation of the [FT_EE_Program](#) function was included to provide support for languages such as LabVIEW where problems can occur when string pointers are contained in a structure.

This function interprets the parameter *pData* as a pointer to a structure of type FT_PROGRAM_DATA that contains the data to write to the EEPROM. The data is written to EEPROM, then read back and verified.

The string pointer parameters in the FT_PROGRAM_DATA structure should be allocated as DWORDs to avoid overlapping of parameters. The string parameters are then passed in separately.

If the *SerialNumber* field is NULL, or *SerialNumber* points to a NULL string, a serial number based on the *ManufacturerId* and the current date and time will be generated. The *Manufacturer* string length plus the *Description* string length must be less than or equal to 40 characters.

Note that the DLL must be informed which version of the FT_PROGRAM_DATA structure is being used. This is done through the *Signature1*, *Signature2* and *Version* elements of the structure. *Signature1* should always be 0x00000000, *Signature2* should always be 0xFFFFFFFF and *Version* can be set to use whichever version is required. For compatibility with all current devices *Version* should be set to the latest version of the FT_PROGRAM_DATA structure which is defined in FTD2XX.h.

If *pData* is NULL, the structure version will default to 0 (original BM series) and the device will be programmed with the default data:

4.8 FT_EE_UASize

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Get the available size of the EEPROM user area.

Definition

FT_STATUS FT_EE_UASizeWrite (FT_HANDLE *ftHandle*, LPDWORD *lpdwSize*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpdwSize</i>	Pointer to a DWORD that receives the available size, in bytes, of the EEPROM user area.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

The user area of an FTDI device EEPROM is the total area of the EEPROM that is unused by device configuration information and descriptors. This area is available to the user to store information specific to their application. The size of the user area depends on the length of the *Manufacturer*, *ManufacturerId*, *Description* and *SerialNumber* strings programmed into the EEPROM.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus = FT_Open(0, &ftHandle);

if (ftStatus != FT_OK) {
    // FT_Open FAILED!
}

DWORD EEUA_Size;

ftStatus = FT_EE_UASize(ftHandle, &EEUA_Size);
if (ftStatus == FT_OK) {
    // FT_EE_UASize OK
    // EEUA_Size contains the size, in bytes, of the EEUA
}
else {
    // FT_EE_UASize FAILED!
}
FT_Close(ftHandle);
```

4.9 FT_EE_UARead

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Read the contents of the EEPROM user area.

Definition

FT_STATUS **FT_EE_UARead** (FT_HANDLE *ftHandle*, PCHAR *pucData*, DWORD *dwDataLen*, LPDWORD *lpdwBytesRead*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>pucData</i>	Pointer to a buffer that contains storage for data to be read.
<i>dwDataLen</i>	Size, in bytes, of buffer that contains storage for the data to be read.
<i>lpdwBytesRead</i>	Pointer to a DWORD that receives the number of bytes read.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function interprets the parameter *pucData* as a pointer to an array of bytes of size *dwDataLen* that contains storage for the data to be read from the EEPROM user area. The actual number of bytes read is stored in the DWORD referenced by *lpdwBytesRead*.

If *dwDataLen* is less than the size of the EEPROM user area, then *dwDataLen* bytes are read into the buffer. Otherwise, the whole of the EEPROM user area is read into the buffer. The available user area size can be determined by calling [FT_EE_UASize](#).

An application should check the function return value and *lpdwBytesRead* when FT_EE_UARead returns.

Example

```
FT_HANDLE ftHandle;  
FT_STATUS ftStatus = FT_Open(0, &ftHandle);  
  
if (ftStatus != FT_OK) {  
    // FT_Open FAILED!  
}  
  
unsigned char Buffer[64];  
DWORD BytesRead;  
  
ftStatus = FT_EE_UARead(ftHandle, Buffer, 64, &BytesRead);  
if (ftStatus == FT_OK) {  
    // FT_EE_UARead OK  
    // User Area data stored in Buffer  
    // Number of bytes read from EEUA stored in BytesRead  
}  
else {  
    // FT_EE_UARead FAILED!  
}  
FT_Close(ftHandle);
```

4.10 FT_EE_UAWrite

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Write data into the EEPROM user area.

Definition

FT_STATUS **FT_EE_UAWrite** (FT_HANDLE *ftHandle*, PCHAR *pucData*, DWORD *dwDataLen*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>pucData</i>	Pointer to a buffer that contains the data to be written.
<i>dwDataLen</i>	Size, in bytes, of buffer that contains storage for the data to be read.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function interprets the parameter *pucData* as a pointer to an array of bytes of size *dwDataLen* that contains the data to be written to the EEPROM user area. It is a programming error for *dwDataLen* to be greater than the size of the EEPROM user area. The available user area size can be determined by calling [FT_EE_UASize](#).

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus = FT_Open(0, &ftHandle);

if (ftStatus != FT_OK) {
    // FT_Open FAILED!
}

char *buffer = "Hello, World";

ftStatus = FT_EE_UAWrite(ftHandle, (unsigned char*)buffer, 12);
if(ftStatus != FT_OK) {
    // FT_EE_UAWRITE failed
}
else {
    // FT_EE_UAWRITE failed
}

FT_Close(ftHandle);
```

5 Extended API Functions

The extended API functions do not apply to FT8U232AM or FT8U245AM devices. FTDI's other USB-UART and USB-FIFO ICs (the FT2232H, FT4232H, FT232R, FT245R, FT2232, FT232B and FT245B) do support these functions. Note that there is device dependence in some of these functions.

5.1 FT_SetLatencyTimer

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Set the latency timer value.

Definition

FT_STATUS **FT_SetLatencyTimer** (FT_HANDLE *ftHandle*, UCHAR *ucTimer*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>ucTimer</i>	Required value, in milliseconds, of latency timer. Valid range is 2 – 255.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

In the FT8U232AM and FT8U245AM devices, the receive buffer timeout that is used to flush remaining data from the receive buffer was fixed at 16 ms. In all other FTDI devices, this timeout is programmable and can be set at 1 ms intervals between 2ms and 255 ms. This allows the device to be better optimized for protocols requiring faster response times from short data packets.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
UCHAR LatencyTimer = 10;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}
```

```
ftStatus = FT_SetLatencyTimer(ftHandle, LatencyTimer);
if (ftStatus == FT_OK) {
    // LatencyTimer set to 10 milliseconds
}
else {
    // FT_SetLatencyTimer FAILED!
}

FT_Close(ftHandle);
```

5.2 FT_GetLatencyTimer

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Get the current value of the latency timer.

Definition

FT_STATUS **FT_GetLatencyTimer** (FT_HANDLE *ftHandle*, PCHAR *pucTimer*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>pucTimer</i>	Pointer to unsigned char to store latency timer value.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

In the FT8U232AM and FT8U245AM devices, the receive buffer timeout that is used to flush remaining data from the receive buffer was fixed at 16 ms. In all other FTDI devices, this timeout is programmable and can be set at 1 ms intervals between 2ms and 255 ms. This allows the device to be better optimized for protocols requiring faster response times from short data packets.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
UCHAR LatencyTimer;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
```

```
        return;
    }

    ftStatus = FT_GetLatencyTimer(ftHandle, &LatencyTimer);
    if (ftStatus == FT_OK) {
        // LatencyTimer contains current value
    }
    else {
        // FT_GetLatencyTimer FAILED!
    }

    FT_Close(ftHandle);
```

5.3 FT_SetBitMode

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Enables different chip modes.

Definition

FT_STATUS **FT_SetBitmode** (FT_HANDLE *ftHandle*, UCHAR *ucMask*, UCHAR *ucMode*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>ucMask</i>	Required value for bit mode mask. This sets up which bits are inputs and outputs. A bit value of 0 sets the corresponding pin to an input, a bit value of 1 sets the corresponding pin to an output. In the case of CBUS Bit Bang, the upper nibble of this value controls which pins are inputs and outputs, while the lower nibble controls which of the outputs are high and low.
<i>ucMode</i>	Mode value. Can be one of the following:
0x0	Reset
0x1	Asynchronous Bit Bang
0x2	MPSSE (FT2232, FT2232H, FT4232H and FT232H devices only)
0x4	Synchronous Bit Bang (FT232R, FT245R, FT2232, FT2232H, FT4232H and FT232H devices only)
0x8	MCU Host Bus Emulation Mode (FT2232, FT2232H, FT4232H and FT232H devices only)
0x10	Fast Opto-Isolated Serial Mode (FT2232, FT2232H, FT4232H and FT232H devices only)
0x20	CBUS Bit Bang Mode (FT232R and FT232H devices only)
0x40	Single Channel Synchronous 245 FIFO Mode (FT2232H and FT232H devices only)

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

For a description of available bit modes for the FT232R, see the application note "Bit Bang Modes for the FT232R and FT245R".

For a description of available bit modes for the FT2232, see the application note "Bit Mode Functions for the FT2232".

For a description of Bit Bang Mode for the FT232B and FT245B, see the application note "FT232B/FT245B Bit Bang Mode".

Application notes are available for download from the FTDI website.

Note that to use CBUS Bit Bang for the FT232R, the CBUS must be configured for CBUS Bit Bang in the EEPROM.

Note that to use Single Channel Synchronous 245 FIFO mode for the FT2232H, channel A must be configured for FT245 FIFO mode in the EEPROM.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
UCHAR Mask = 0xff;
UCHAR Mode = 1; // Set asynchronous bit-bang mode

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}

ftStatus = FT_SetBitMode(ftHandle, Mask, Mode);
if (ftStatus == FT_OK) {
    // 0xff written to device
}
else {
    // FT_SetBitMode FAILED!
}

FT_Close(ftHandle);
```

5.4 FT_GetBitMode

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Gets the instantaneous value of the data bus.

Definition

FT_STATUS **FT_GetBitmode** (FT_HANDLE *ftHandle*, PCHAR *pucMode*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>pucMode</i>	Pointer to unsigned char to store the instantaneous data bus value.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

For a description of available bit modes for the FT232R, see the application note "Bit Bang Modes for the FT232R and FT245R".

For a description of available bit modes for the FT2232, see the application note "Bit Mode Functions for the FT2232".

For a description of bit bang modes for the FT232B and FT245B, see the application note "FT232B/FT245B Bit Bang Mode".

For a description of bit modes supported by the FT4232H and FT2232H devices, please see the IC data sheets.

These application notes are available for download from the FTDI website.

Example

```
FT_HANDLE ftHandle;
UCHAR BitMode;
FT_STATUS ftStatus;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}

ftStatus = FT_GetBitMode(ftHandle, &BitMode);
if (ftStatus == FT_OK) {
    // BitMode contains current value
}
else {
    // FT_GetBitMode FAILED!
}

FT_Close(ftHandle);
```

5.5 FT_SetUSBParameters

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Set the USB request transfer size.

Definition

FT_STATUS **FT_SetUSBParameters** (FT_HANDLE *ftHandle*, DWORD *dwInTransferSize*, DWORD *dwOutTransferSize*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>dwInTransferSize</i>	Transfer size for USB IN request.
<i>dwOutTransferSize</i>	Transfer size for USB OUT request.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function can be used to change the transfer sizes from the default transfer size of 4096 bytes to better suit the application requirements. Transfer sizes must be set to a multiple of 64 bytes between 64 bytes and 64k bytes.

When FT_SetUSBParameters is called, the change comes into effect immediately and any data that was held in the driver at the time of the change is lost.

Note that, at present, only *dwInTransferSize* is supported.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
DWORD InTransferSize = 16384;

ftStatus = FT_Open(0, &ftHandle);
if(ftStatus != FT_OK) {
    // FT_Open failed
    return;
}

ftStatus = FT_SetUSBParameters(ftHandle, InTransferSize, 0);
if (ftStatus == FT_OK) {
    // In transfer size set to 16 Kbytes
}
else {
    // FT_SetUSBParameters FAILED!
}

FT_Close(ftHandle);
```

6 FT-Win32 API Functions

The functions in this section are supplied to ease porting from a Win32 serial port application. These functions are supported under non-Windows platforms to assist with porting existing applications from Windows. Note that classic D2XX functions and the Win32 D2XX functions should not be mixed unless stated.

6.1 FT_W32_CreateFile

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Opens the specified device and return a handle which will be used for subsequent accesses. The device can be specified by its serial number, device description, or location.

This function must be used if overlapped I/O is required.

Definition

FT_HANDLE **FT_W32_CreateFile** (PVOID *pvArg1*, DWORD *dwAccess*, DWORD *dwShareMode*, LPSECURITY_ATTRIBUTES *lpSecurityAttributes*, DWORD *dwCreate*, DWORD *dwAttrsAndFlags*, HANDLE *hTemplate*)

Parameters

pvArg1 Meaning depends on the value of *dwAttrsAndFlags*. Can be a pointer to a null terminated string that contains the description or serial number of the device, or can be the location of the device. These values can be obtained from the [FT_CreateDeviceInfoList](#), [FT_GetDeviceInfoDetail](#) or [FT_ListDevices](#) functions.

<i>dwAccess</i>	Type of access to the device. Access can be GENERIC_READ, GENERIC_WRITE or both. Ignored in Linux.
<i>dwShareMode</i>	How the device is shared. This value must be set to 0.
<i>lpSecurityAttributes</i>	This parameter has no effect and should be set to NULL.
<i>dwCreate</i>	This parameter must be set to OPEN_EXISTING. Ignored in Linux.
<i>dwAttrsAndFlags</i>	File attributes and flags. This parameter is a combination of FILE_ATTRIBUTE_NORMAL, FILE_FLAG_OVERLAPPED if overlapped I/O is used, FT_OPEN_BY_SERIAL_NUMBER if <i>lp.szName</i> is the device's serial number, and FT_OPEN_BY_DESCRIPTION if <i>lp.szName</i> is the device's description.
<i>hTemplate</i>	This parameter must be NULL.

Return Value

If the function is successful, the return value is a handle.

If the function is unsuccessful, the return value is the Win32 error code INVALID_HANDLE_VALUE.

Remarks

The meaning of pvArg1 depends on dwAttrsAndFlags: if [FT_OPEN_BY_SERIAL_NUMBER](#) or [FT_OPEN_BY_DESCRIPTION](#) is set in dwAttrsAndFlags, pvArg1 contains a pointer to a null terminated string that contains the device's serial number or description; if [FT_OPEN_BY_LOCATION](#) is set in dwAttrsAndFlags, pvArg1 is interpreted as a value of type long that contains the location ID of the device.

dwAccess can be GENERIC_READ, GENERIC_WRITE or both; dwShareMode must be set to 0; lpSecurityAttributes must be set to NULL; dwCreate must be set to OPEN_EXISTING; dwAttrsAndFlags is a combination of FILE_ATTRIBUTE_NORMAL, FILE_FLAG_OVERLAPPED if overlapped I/O is used, [FT_OPEN_BY_SERIAL_NUMBER](#) or [FT_OPEN_BY_DESCRIPTION](#) or [FT_OPEN_BY_LOCATION](#); hTemplate must be NULL.

Note that Linux, Mac OS X and Windows CE do not support overlapped IO or location IDs.

Examples

The examples that follow use these variables.

```
FT_STATUS ftStatus;  
FT_HANDLE ftHandle;  
char Buf[64];
```

1. Open a device for overlapped I/O using its serial number

```
ftStatus = FT_ListDevices(0, Buf, FT_LIST_BY_INDEX | FT_OPEN_BY_SERIAL_NUMBER);  
  
ftHandle = FT_W32_CreateFile(Buf, GENERIC_READ | GENERIC_WRITE, 0, 0,  
                             OPEN_EXISTING,  
                             FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED |  
                             FT_OPEN_BY_SERIAL_NUMBER,  
                             0);  
  
if (ftHandle == INVALID_HANDLE_VALUE)  
    ; // FT_W32_CreateDevice failed
```

2. Open a device for non-overlapped I/O using its description

```
ftStatus = FT_ListDevices(0, Buf, FT_LIST_BY_INDEX | FT_OPEN_BY_DESCRIPTION);  
  
ftHandle = FT_W32_CreateFile(Buf, GENERIC_READ | GENERIC_WRITE, 0, 0,  
                             OPEN_EXISTING,  
                             FILE_ATTRIBUTE_NORMAL | FT_OPEN_BY_DESCRIPTION,  
                             0);  
  
if (ftHandle == INVALID_HANDLE_VALUE)  
    ; // FT_W32_CreateDevice failed
```

3. Open a device for non-overlapped I/O using its location

```
long locID;  
  
ftStatus = FT_ListDevices(0, &locID, FT_LIST_BY_INDEX | FT_OPEN_BY_LOCATION);  
  
ftHandle = FT_W32_CreateFile((PVOID) locID, GENERIC_READ | GENERIC_WRITE, 0, 0,  
                             OPEN_EXISTING,  
                             FILE_ATTRIBUTE_NORMAL | FT_OPEN_BY_LOCATION,  
                             0);  
  
if (ftHandle == INVALID_HANDLE_VALUE)
```

```
; // FT_W32_CreateDevice failed
```

6.2 FT_W32_CloseHandle

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Close the specified device handle.

Definition

BOOL **FT_W32_CloseHandle** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Example

This example shows how to close a device after opening it for non-overlapped I/O using its description.

```
FT_STATUS ftStatus;  
FT_HANDLE ftHandle;  
char Buf[64];  
  
ftStatus = FT_ListDevices(0, Buf, FT_LIST_BY_INDEX|FT_OPEN_BY_DESCRIPTION);  
ftHandle = FT_W32_CreateFile (Buf, GENERIC_READ|GENERIC_WRITE, 0, 0,  
                             OPEN_EXISTING,  
                             FILE_ATTRIBUTE_NORMAL | FT_OPEN_BY_DESCRIPTION,  
                             0);  
if (ftHandle == INVALID_HANDLE_VALUE){  
    // FT_W32_CreateDevice failed  
}  
else {  
    // FT_W32_CreateFile OK, so do some work, and eventually ...  
    FT_W32_CloseHandle(ftHandle);  
}
```

6.3 FT_W32_ReadFile

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)
Windows CE (4.2 and later)

Summary

Read data from the device.

Definition

BOOL **FT_W32_ReadFile** (FT_HANDLE *ftHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToRead*, LPDWORD *lpdwBytesReturned*, LPOVERLAPPED *lpOverlapped*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpBuffer</i>	Pointer to a buffer that receives the data from the device.
<i>dwBytesToRead</i>	Number of bytes to read from the device.
<i>lpdwBytesReturned</i> the device.	Pointer to a variable that receives the number of bytes read from
<i>lpOverlapped</i>	Pointer to an overlapped structure.

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Remarks

This function supports both non-overlapped and overlapped I/O, except under Linux, Mac OS X and Windows CE where only non-overlapped IO is supported.

Non-overlapped I/O

The parameter, *lpOverlapped*, must be NULL for non-overlapped I/O.

This function always returns the number of bytes read in *lpdwBytesReturned*.

This function does not return until *dwBytesToRead* have been read into the buffer. The number of bytes in the receive queue can be determined by calling [FT_GetStatus](#) or [FT_GetQueueStatus](#), and passed as *dwBytesToRead* so that the function reads the device and returns immediately.

When a read timeout has been setup in a previous call to [FT_W32_SetCommTimeouts](#), this function returns when the timer expires or *dwBytesToRead* have been read, whichever occurs first. If a timeout occurred, any available data is read into *lpBuffer* and the function returns a non-zero value.

An application should use the function return value and *lpdwBytesReturned* when processing the buffer. If the return value is non-zero and *lpdwBytesReturned* is equal to *dwBytesToRead* then the function has completed normally. If the return value is non-zero and *lpdwBytesReturned* is less than *dwBytesToRead* then a timeout has occurred, and the read request has been partially completed. Note that if a timeout occurred and no data was read, the return value is still non-zero.

A return value of *FT_IO_ERROR* suggests an error in the parameters of the function, or a fatal error like USB disconnect has occurred.

Overlapped I/O

When the device has been opened for overlapped I/O, an application can issue a request and perform some additional work while the request is pending. This contrasts with the case of non-overlapped I/O in which the application issues a request and receives control again only after the request has been completed.

The parameter, *lpOverlapped*, must point to an initialized OVERLAPPED structure.

If there is enough data in the receive queue to satisfy the request, the request completes immediately and the return code is non-zero. The number of bytes read is returned in *lpdwBytesReturned*.

If there is not enough data in the receive queue to satisfy the request, the request completes immediately, and the return code is zero, signifying an error. An application should call [FT_W32_GetLastError](#) to get the cause of the error. If the error code is ERROR_IO_PENDING, the overlapped operation is still in progress, and the application can perform other processing. Eventually, the application checks the result of the overlapped request by calling [FT_W32_GetOverlappedResult](#).

If successful, the number of bytes read is returned in *lpdwBytesReturned*.

Example

1. This example shows how to read 256 bytes from the device using non-overlapped I/O.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile for non-overlapped i/o
char Buf[256];
DWORD dwToRead = 256;
DWORD dwRead;

if (FT_W32_ReadFile(ftHandle, Buf, dwToRead, &dwRead, &osRead)) {
    if (dwToRead == dwRead) {
        // FT_W32_ReadFile OK
    }
    else {
        // FT_W32_ReadFile timeout
    }
}
else {
    // FT_W32_ReadFile failed
}
```

2. This example shows how to read 256 bytes from the device using overlapped I/O.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile for overlapped i/o
char Buf[256];
DWORD dwToRead = 256;
DWORD dwRead;
OVERLAPPED osRead = { 0 };
osRead.hEvent = CreateEvent (NULL, FALSE, FALSE, NULL);

if (!FT_W32_ReadFile(ftHandle, Buf, dwToRead, &dwRead, &osRead)) {
    if (FT_W32_GetLastError(ftHandle) == ERROR_IO_PENDING) {
        // write is delayed so do some other stuff until ...
        if (!FT_W32_GetOverlappedResult(ftHandle, &osRead, &dwRead, FALSE)) {
            // error
        }
        else {
            if (dwToRead == dwRead) {
                // FT_W32_ReadFile OK
            }
            else {
                // FT_W32_ReadFile timeout
            }
        }
    }
}
else {
    // FT_W32_ReadFile OK
}
```

```
CloseHandle (osRead.hEvent);
```

6.4 FT_W32_WriteFile

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Write data to the device.

Definition

BOOL **FT_W32_WriteFile** (FT_HANDLE *ftHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToWrite*, LPDWORD *lpdwBytesWritten*, LPOVERLAPPED *lpOverlapped*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpBuffer</i>	Pointer to the buffer that contains the data to write to the device.
<i>dwBytesToWrite</i>	Number of bytes to be written to the device.
<i>lpdwBytesWritten</i>	Pointer to a variable that receives the number of bytes written to the device.
<i>lpOverlapped</i>	Pointer to an overlapped structure.

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Remarks

This function supports both non-overlapped and overlapped I/O, except under Linux, Mac OS X and Windows CE where only non-overlapped IO is supported.

Non-overlapped I/O

The parameter, *lpOverlapped*, must be NULL for non-overlapped I/O.

This function always returns the number of bytes written in *lpdwBytesWritten*.

This function does not return until *dwBytesToWrite* have been written to the device.

When a write timeout has been setup in a previous call to [FT_W32_SetCommTimeouts](#), this function returns when the timer expires or *dwBytesToWrite* have been written, whichever occurs first. If a timeout occurred, *lpdwBytesWritten* contains the number of bytes actually written, and the function returns a non-zero value.

An application should always use the function return value and *lpdwBytesWritten*. If the return value is non-zero and *lpdwBytesWritten* is equal to *dwBytesToWrite* then the function has completed normally. If the return value is non-zero and *lpdwBytesWritten* is less than *dwBytesToWrite* then a timeout has occurred, and the write request has been partially completed. Note that if a timeout occurred and no data was written, the return value is still non-zero.

Overlapped I/O

When the device has been opened for overlapped I/O, an application can issue a request and perform some additional work while the request is pending. This contrasts with the case of non-overlapped I/O in which the application issues a request and receives control again only after the request has been completed.

The parameter, *lpOverlapped*, must point to an initialized OVERLAPPED structure.

This function completes immediately, and the return code is zero, signifying an error. An application should call [FT_W32_GetLastError](#) to get the cause of the error. If the error code is ERROR_IO_PENDING, the overlapped operation is still in progress, and the application can perform other processing. Eventually, the application checks the result of the overlapped request by calling [FT_W32_GetOverlappedResult](#).

If successful, the number of bytes written is returned in *lpdwBytesWritten*.

Example

1. This example shows how to write 128 bytes to the device using non-overlapped I/O.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile for overlapped i/o
char Buf[128]; // contains data to write to the device
DWORD dwToWrite = 128;
DWORD dwWritten;

if (FT_W32_WriteFile(ftHandle, Buf, dwToWrite, &dwWritten, &osWrite)) {
    if (dwToWrite == dwWritten){
        // FT_W32_WriteFile OK
    }
    else{
        // FT_W32_WriteFile timeout
    }
}
else{
    // FT_W32_WriteFile failed
}
```

2. This example shows how to write 128 bytes to the device using overlapped I/O.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile for overlapped i/o
char Buf[128]; // contains data to write to the device
DWORD dwToWrite = 128;
DWORD dwWritten;
OVERLAPPED osWrite = { 0 };

if (!FT_W32_WriteFile(ftHandle, Buf, dwToWrite, &dwWritten, &osWrite)) {
    if (FT_W32_GetLastError(ftHandle) == ERROR_IO_PENDING) {
        // write is delayed so do some other stuff until ...
        if (!FT_W32_GetOverlappedResult(ftHandle, &osWrite, &dwWritten, FALSE)){
            // error
        }
        else {
            if (dwToWrite == dwWritten){
                // FT_W32_WriteFile OK
            }
            else{
                // FT_W32_WriteFile timeout
            }
        }
    }
}
```

```
        }  
    }  
    else {  
        // FT_W32_WriteFile OK  
    }  
}
```

6.5 FT_W32_GetOverlappedResult

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

Gets the result of an overlapped operation.

Definition

BOOL **FT_W32_GetOverlappedResult** (FT_HANDLE *ftHandle*, LPOVERLAPPED *lpOverlapped*, LPDWORD *lpdwBytesTransferred*, BOOL *bWait*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpOverlapped</i>	Pointer to an overlapped structure.
<i>lpdwBytesTransferred</i>	Pointer to a variable that receives the number of bytes transferred during the overlapped operation.
<i>bWait</i>	Set to TRUE if the function does not return until the operation has been completed.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Remarks

This function is used with overlapped I/O and so is not supported in Linux, Mac OS X or Windows CE. For a description of its use, see [FT_W32_ReadFile](#) and [FT_W32_WriteFile](#).

6.6 FT_W32_EscapeCommFunction

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
~~Windows (2000 and later)~~

Windows CE (4.2 and later)

Summary

Perform an extended function.

Definition

BOOL **FT_W32_EscapeCommFunction** (FT_HANDLE *ftHandle*, DWORD *dwFunc*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>dwFunc</i>	The extended function to perform can be one of the following values: - CLRDTR – Clear the DTR signal - CLRRTS – Clear the RTS signal - SETDTR – Set the DTR signal - SETRTS – Set the RTS signal - SETBREAK – Set the BREAK condition - CLRBREAK – Clear the BREAK condition

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile

FT_W32_EscapeCommFunction(ftHandle,CLRDTR); // Clear the DTR signal
FT_W32_EscapeCommFunction(ftHandle,SETRTS); // Set the RTS signal
```

6.7 FT_W32_GetCommModemStatus

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function gets the current modem control value.

Definition

BOOL **FT_W32_GetCommModemStatus** (FT_HANDLE *ftHandle*, LPDWORD *lpdwStat*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpdwStat</i>	Pointer to a variable to contain modem control value. The modem control value can be a combination of the following: - MS_CTS_ON – Clear To Send (CTS) is on - MS_DSR_ON – Data Set Ready (DSR) is on - MS_RING_ON – Ring Indicator (RI) is on - MS_RLSD_ON – Receive Line Signal Detect (RLSD) is on

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile DWORD dwStatus;

if (FT_W32_GetCommModemStatus(ftHandle,&dwStatus)) {
    // FT_W32_GetCommModemStatus ok
    if (dwStatus & MS_CTS_ON)
        ; // CTS is on
    if (dwStatus & MS_DSR_ON)
        ; // DSR is on
    if (dwStatus & MS_RI_ON)
        ; // RI is on
    if (dwStatus & MS_RLSD_ON)
        ; // RLSD is on
}
else
    ; // FT_W32_GetCommModemStatus failed
```

6.8 FT_W32_SetupComm

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

This function sets the read and write buffers.

Definition

BOOL **FT_W32_SetupComm** (FT_HANDLE *ftHandle*, DWORD *dwReadBufferSize*,
DWORD *dwWriteBufferSize*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>dwReadBufferSize</i>	Length, in bytes, of the read buffer.
<i>dwWriteBufferSize</i>	Length, in bytes, of the write buffer.

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Remarks

This function has no effect. It is the responsibility of the driver to allocate sufficient storage for I/O requests.

6.9 FT_W32_SetCommState

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function sets the state of the device according to the contents of a device control block (DCB).

Definition

BOOL **FT_W32_SetCommState** (FT_HANDLE *ftHandle*, LPFTDCB *lpftDcb*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpftDcb</i>	Pointer to an FTDCB structure.

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
FTDCB ftDCB;

if (FT_W32_GetCommState(ftHandle, &ftDCB)) {
    // FT_W32_GetCommState ok, device state is in ftDCB
    ftDCB.BaudRate = 921600; // Change the baud rate
}
```

```
    if (FT_W32_SetCommState(ftHandle,&ftDCB))  
        ; // FT_W32_SetCommState ok  
    else  
        ; // FT_W32_SetCommState failed  
}  
else  
    ; // FT_W32_GetCommState failed
```

6.10 FT_W32_GetCommState

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function gets the current device state.

Definition

BOOL **FT_W32_GetCommState** (FT_HANDLE *ftHandle*, LPFTDCB *lpftDcb*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpftDcb</i>	Pointer to an FTDCB structure.

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Remarks

The current state of the device is returned in a device control block.

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile  
FTDCB ftDCB;  
  
if (FT_W32_GetCommState(ftHandle,&ftDCB))  
    ; // FT_W32_GetCommState ok, device state is in ftDCB  
else  
    ; // FT_W32_GetCommState failed
```

6.11 FT_W32_SetCommTimeouts

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

This function sets the timeout parameters for I/O requests.

Definition

BOOL **FT_W32_SetCommTimeouts** (FT_HANDLE *ftHandle*, LPFTTIMEOUTS *lpftTimeouts*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpftTimeouts</i>	Pointer to an FTTIMEOUTS structure to store timeout information.

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Remarks

Timeouts are calculated using the information in the FTTIMEOUTS structure.

For read requests, the number of bytes to be read is multiplied by the total timeout multiplier, and added to the total timeout constant. So, if TS is an FTTIMEOUTS structure and the number of bytes to read is *dwToRead*, the read timeout, *rdTO*, is calculated as follows.

$$rdTO = (dwToRead * TS.ReadTotalTimeoutMultiplier) + TS.ReadTotalTimeoutConstant$$

For write requests, the number of bytes to be written is multiplied by the total timeout multiplier, and added to the total timeout constant. So, if TS is an FTTIMEOUTS structure and the number of bytes to write is *dwToWrite*, the write timeout, *wrTO*, is calculated as follows.

$$wrTO = (dwToWrite * TS.WriteTotalTimeoutMultiplier) + TS.WriteTotalTimeoutConstant$$

Linux and Mac OS X currently ignore the *ReadIntervalTimeout*, *ReadTotalTimeoutMultiplier* and *WriteTotalTimeoutMultiplier*.

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
FTTIMEOUTS ftTS;

ftTS.ReadIntervalTimeout = 0;
ftTS.ReadTotalTimeoutMultiplier = 0;
ftTS.ReadTotalTimeoutConstant = 100;
ftTS.WriteTotalTimeoutMultiplier = 0;
ftTS.WriteTotalTimeoutConstant = 200;

if (FT_W32_SetCommTimeouts(ftHandle, &ftTS))
    ; // FT_W32_SetCommTimeouts OK
```

```
else  
    ; // FT_W32_SetCommTimeouts failed
```

6.12 FT_W32_GetCommTimeouts

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function gets the current read and write request timeout parameters for the specified device.

Definition

BOOL **FT_W32_GetCommTimeouts** (FT_HANDLE *ftHandle*, LPFTTIMEOUTS *lpftTimeouts*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpftTimeouts</i>	Pointer to an FTTIMEOUTS structure to store timeout information.

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Remarks

For an explanation of how timeouts are used, see [FT_W32_SetCommTimeouts](#).

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile  
FTTIMEOUTS ftTS;  
  
if (FT_W32_GetCommTimeouts(ftHandle,&ftTS))  
    ; // FT_W32_GetCommTimeouts OK  
else  
    ; // FT_W32_GetCommTimeouts failed
```

6.13 FT_W32_SetCommBreak

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)
Windows CE (4.2 and later)

Summary

Puts the communications line in the BREAK state.

Definition

BOOL **FT_W32_SetCommBreak** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

If the function is successful, the return value is nonzero.
If the function is unsuccessful, the return value is zero.

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile

if (!FT_W32_SetCommBreak(ftHandle))
    ; // FT_W32_SetCommBreak failed
else
    ; // FT_W32_SetCommBreak OK
```

6.14 FT_W32_ClearCommBreak

Supported Operating Systems

Linux
Mac OS X (10.4 and later)
Windows (2000 and later)
Windows CE (4.2 and later)

Summary

Puts the communications line in the non-BREAK state.

Definition

BOOL **FT_W32_ClearCommBreak** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile

if (!FT_W32_ClearCommBreak(ftHandle)){
    // FT_W32_ClearCommBreak failed
}
else{
    // FT_W32_ClearCommBreak OK
}
```

6.15 FT_W32_SetCommMask

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function specifies events that the device has to monitor.

Definition

BOOL **FT_W32_SetCommMask** (FT_HANDLE *ftHandle*, DWORD *dwMask*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>dwMask</i>	Mask containing events that the device has to monitor. This can be a combination of the following: - EV_BREAK – BREAK condition detected - EV_CTS – Change in Clear To Send (CTS) - EV_DSR – Change in Data Set Ready (DSR) - EV_ERR – Error in line status - EV_RING – Change in Ring Indicator (RI) - EV_RLSD – Change in Receive Line Signal Detect (RLSD) - EV_RXCHAR – Character received - EV_RXFLAG – Event character received - EV_TXEMPTY – Transmitter empty

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Remarks

This function specifies the events that the device should monitor. An application can call the function [FT_W32_WaitCommEvent](#) to wait for an event to occur.

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
DWORD dwMask = EV_CTS | EV_DSR;

if (!FT_W32_SetCommMask(ftHandle, dwMask))
    ; // FT_W32_SetCommMask failed
else
    ; // FT_W32_SetCommMask OK
```

6.16 FT_W32_GetCommMask

Supported Operating Systems

Windows (2000 and later)

Summary

Retrieves the events that are currently being monitored by a device.

Definition

BOOL **FT_W32_GetCommMask** (FT_HANDLE *ftHandle*, LPDWORD *lpdwEventMask*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpdwEventMask</i>	Pointer to a location that receives a mask that contains the events that are currently enabled. This parameter can be one or more of the following values: - EV_BREAK – BREAK condition detected - EV_CTS – Change in Clear To Send (CTS) - EV_DSR – Change in Data Set Ready (DSR) - EV_ERR – Error in line status - EV_RING – Change in Ring Indicator (RI) - EV_RLSD – Change in Receive Line Signal Detect (RLSD) - EV_RXCHAR – Character received - EV_RXFLAG – Event character received - EV_TXEMPTY – Transmitter empty

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Remarks

This function returns events currently being monitored by the device. Event monitoring for these events is enabled by the [FT_W32_SetCommMask](#) function.

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
DWORD dwMask;

if (!FT_W32_GetCommMask(ftHandle, &dwMask))
    ; // FT_W32_GetCommMask failed
else
    ; // FT_W32_GetCommMask OK
```

6.17 FT_W32_WaitCommEvent

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function waits for an event to occur.

Definition

BOOL **FT_W32_SetupComm** (FT_HANDLE *ftHandle*, LPDWORD *lpdwEvent*,
LPOVERLAPPED *lpOverlapped*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpdwEvent</i>	Pointer to a location that receives a mask that contains the events that occurred.
<i>lpOverlapped</i>	Pointer to an overlapped structure.

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Remarks

This function supports both non-overlapped and overlapped I/O, except under Windows CE and Linux where only non-overlapped I/O is supported.

Non-overlapped I/O

The parameter, *lpOverlapped*, must be NULL for non-overlapped I/O.

This function does not return until an event that has been specified in a call to [FT_W32_SetCommMask](#) has occurred. The events that occurred and resulted in this function returning are stored in *lpdwEvent*.

Overlapped I/O

When the device has been opened for overlapped I/O, an application can issue a request and perform some additional work while the request is pending. This contrasts with the case of non-overlapped I/O in which the application issues a request and receives control again only after the request has been completed.

The parameter, *lpOverlapped*, must point to an initialized OVERLAPPED structure.

This function does not return until an event that has been specified in a call to [FT_W32_SetCommMask](#) has occurred.

If an event has already occurred, the request completes immediately, and the return code is non-zero. The events that occurred are stored in *lpdwEvent*.

If an event has not yet occurred, the request completes immediately, and the return code is zero, signifying an error. An application should call [FT_W32_GetLastError](#) to get the cause of the error. If the error code is ERROR_IO_PENDING, the overlapped operation is still in progress, and the application can perform other processing. Eventually, the application checks the result of the overlapped request by calling [FT_W32_GetOverlappedResult](#). The events that occurred and resulted in this function returning are stored in *lpdwEvent*.

Examples

1. This example shows how to write 128 bytes to the device using non-overlapped I/O.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile for non-overlapped i/o
DWORD dwEvents;

if (FT_W32_WaitCommEvent(ftHandle, &dwEvents, NULL))
    ; // FT_W32_WaitCommEvents OK
else
    ; // FT_W32_WaitCommEvents failed
```

2. This example shows how to write 128 bytes to the device using overlapped I/O.

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile for overlapped i/o
DWORD dwEvents;
DWORD dwRes;
OVERLAPPED osWait = { 0 };

if (!FT_W32_WaitCommEvent(ftHandle, &dwEvents, &osWait)) {
    if (FT_W32_GetLastError(ftHandle) == ERROR_IO_PENDING) {
        // wait is delayed so do some other stuff until ...
        if (!FT_W32_GetOverlappedResult(ftHandle, &osWait, &dwRes, FALSE))
            ; // error
        else
            ; // FT_W32_WaitCommEvent OK
            // Events that occurred are stored in dwEvents
    }
}
else {
    // FT_W32_WaitCommEvent OK
    // Events that occurred are stored in dwEvents
}
```

6.18 FT_W32_PurgeComm

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

This function purges the device.

Definition

BOOL **FT_W32_PurgeComm** (FT_HANDLE *ftHandle*, DWORD *dwFlags*)

Parameters

ftHandle

Handle of the device.

dwFlags

Specifies the action to take. The action can be a combination of the following:

PURGE_TXABORT – Terminate outstanding overlapped writes

PURGE_RXABORT – Terminate outstanding overlapped reads

PURGE_TXCLEAR – Clear the transmit buffer

PURGE_RXCLEAR – Clear the receive buffer

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile

if (FT_W32_PurgeComm(ftHandle, PURGE_TXCLEAR|PURGE_RXCLEAR))
    ; // FT_W32_PurgeComm OK
else
    ; // FT_W32_PurgeComm failed
```

6.19 FT_W32_GetLastError

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Gets the last error that occurred on the device.

Definition

DWORD **FT_W32_GetLastError** (FT_HANDLE *ftHandle*)

Parameters

ftHandle Handle of the device.

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Remarks

This function is normally used with overlapped I/O and so is not supported in Windows CE. For a description of its use, see [FT_W32_ReadFile](#) and [FT_W32_WriteFile](#).

In Linux and Mac OS X, this function returns a DWORD that directly maps to the FT Errors (for example the FT_INVALID_HANDLE error number).

6.20 FT_W32_ClearCommError

Supported Operating Systems

Linux

Mac OS X (10.4 and later)

Windows (2000 and later)

Windows CE (4.2 and later)

Summary

Gets information about a communications error and get current status of the device.

Definition

BOOL **FT_W32_ClearCommError** (FT_HANDLE *ftHandle*, LPDWORD *lpdwErrors*, LPFTCOMSTAT *lpftComstat*)

Parameters

<i>ftHandle</i>	Handle of the device.
<i>lpdwErrors</i>	Variable that contains the error mask.
<i>lpftComstat</i>	Pointer to FTCOMSTAT structure.

Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

Example

```
static COMSTAT oldCS = {0};
static DWORD dwOldErrors = 0;

FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
COMSTAT newCS;
DWORD dwErrors;
BOOL bChanged = FALSE;

if (!FT_W32_ClearCommError(ftHandle, &dwErrors, (FTCOMSTAT *)&newCS))
    ; // FT_W32_ClearCommError failed

if (dwErrors != dwOldErrors) {
    bChanged = TRUE;
    dwErrorsOld = dwErrors;
}

if (memcmp(&oldCS, &newCS, sizeof(FTCOMSTAT))) {
    bChanged = TRUE;
    oldCS = newCS;
}

if (bChanged) {
    if (dwErrors & CE_BREAK)
        ; // BREAK condition detected
    if (dwErrors & CE_FRAME)
        ; // Framing error detected
    if (dwErrors & CE_RXOVER)
        ; // Receive buffer has overflowed
    if (dwErrors & CE_TXFULL)
        ; // Transmit buffer full
    if (dwErrors & CE_OVERRUN)
        ; // Character buffer overrun
    if (dwErrors & CE_RXPARITY)
        ; // Parity error detected
    if (newCS.fCtsHold)
        ; // Transmitter waiting for CTS
    if (newCS.fDsrHold)
        ; // Transmitter is waiting for DSR
    if (newCS.fRltdHold)
        ; // Transmitter is waiting for RLSD
    if (newCS.fXoffHold)
        ; // Transmitter is waiting because XOFF was received
```



```
    if (newCS.fXoffSent)
        ; //
    if (newCS.fEof)
        ; // End of file character has been received
    if (newCS.fTxim)
        ; // Tx immediate character queued for transmission
    // newCS.cbInQue contains number of bytes in receive queue
    // newCS.cbOutQue contains number of bytes in transmit queue
}
```

7 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place,
Centurion Business Park
Glasgow, G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales)	sales1@ftdichip.com
E-mail (Support)	support1@ftdichip.com
E-mail (General Enquiries)	admin1@ftdichip.com
Web Site URL	http://www.ftdichip.com
Web Shop URL	http://www.ftdichip.com

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
2F, No 516, Sec. 1 NeiHu Road
Taipei 114
Taiwan, R.O.C.
Tel: +886 (0) 2 8791 3570
Fax: +886 (0) 2 8791 3576

E-mail (Sales)	tw.sales1@ftdichip.com
E-mail (Support)	tw.support1@ftdichip.com
E-mail (General Enquiries)	tw.admin1@ftdichip.com
Web Site URL	http://www.ftdichip.com

Branch Office – Hillsboro, Oregon, USA

Future Technology Devices International Limited (USA)
7235 NW Evergreen Parkway, Suite 600
Hillsboro, OR 97123-5803
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales)	us.sales@ftdichip.com
E-Mail (Support)	us.support@ftdichip.com
E-Mail (General Enquiries)	us.admin@ftdichip.com
Web Site URL	http://www.ftdichip.com

Branch Office – Shanghai, China

Future Technology Devices International Limited (China)
Room 408, 317 Xianxia Road,
ChangNing District,
ShangHai, China

Tel: +86 (21) 62351596
Fax: +86(21) 62351595

E-Mail (Sales):	cn.sales@ftdichip.com
E-Mail (Support):	cn.support@ftdichip.com
E-Mail (General Enquiries):	cn.admin1@ftdichip.com
Web Site URL	http://www.ftdichip.com

Distributor and Sales Representatives

Please visit the Sales Network page of the FTDI Web site for the contact details of our distributor(s) and sales representative(s) in your country.

System and equipment manufacturers and designers are responsible to ensure that their systems, and any Future Technology Devices International Ltd (FTDI) devices incorporated in their systems, meet all applicable safety, regulatory and system-level performance requirements. All application-related information in this document (including application descriptions, suggested FTDI devices and other materials) is provided for reference only. While FTDI has taken care to assure it is accurate, this information is subject to customer confirmation, and FTDI disclaims all liability for system designs and for any applications assistance provided by FTDI. Use of FTDI devices in life support and/or safety applications is entirely at the user's risk, and the user agrees to defend, indemnify and hold harmless FTDI from any and all damages, claims, suits or expense resulting from such use. This document is subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Neither the whole nor any part of the information contained in, or the product described in this document, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. Future Technology Devices International Ltd, Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH, United Kingdom. Scotland Registered Company Number: SC136640

8 Appendix A - Type Definitions

UCHAR Unsigned char (1 byte)
e)
PUCHAR Pointer to unsigned char (4 bytes)
PCHAR Pointer to char (4 bytes)
DWORD Unsigned long (4 bytes)
LPDWORD Pointer to unsigned long (4 bytes)
FT_HANDLE DWORD

FT_STATUS (DWORD)

FT_OK = 0
FT_INVALID_HANDLE = 1
FT_DEVICE_NOT_FOUND = 2
FT_DEVICE_NOT_OPENED = 3
FT_IO_ERROR = 4
FT_INSUFFICIENT_RESOURCES = 5
FT_INVALID_PARAMETER = 6
FT_INVALID_BAUD_RATE = 7
FT_DEVICE_NOT_OPENED_FOR_ERASE = 8
FT_DEVICE_NOT_OPENED_FOR_WRITE = 9
FT_FAILED_TO_WRITE_DEVICE = 10
FT_EEPROM_READ_FAILED = 11
FT_EEPROM_WRITE_FAILED = 12
FT_EEPROM_ERASE_FAILED = 13
FT_EEPROM_NOT_PRESENT = 14
FT_EEPROM_NOT_PROGRAMMED = 15
FT_INVALID_ARGS = 16
FT_NOT_SUPPORTED = 17
FT_OTHER_ERROR = 18

Flags (see FT_ListDevices)

FT_LIST_NUMBER_ONLY = 0x80000000
FT_LIST_BY_INDEX = 0x40000000
FT_LIST_ALL = 0x20000000

Flags (see FT_OpenEx)

FT_OPEN_BY_SERIAL_NUMBER = 1
FT_OPEN_BY_DESCRIPTION = 2
FT_OPEN_BY_LOCATION = 4

FT_DEVICE (DWORD)

FT_DEVICE_232BM = 0
FT_DEVICE_232AM = 1
FT_DEVICE_100AX = 2
FT_DEVICE_UNKNOWN = 3
FT_DEVICE_2232C = 4
FT_DEVICE_232R = 5
FT_DEVICE_2232H = 6
FT_DEVICE_4232H = 7
FT_DEVICE_232H = 8

Word Length (see FT_SetDataCharacteristics)

FT_BITS_8 = 8
FT_BITS_7 = 7

Stop Bits (see FT_SetDataCharacteristics)

FT_STOP_BITS_1 = 0

FT_STOP_BITS_2 = 2

Parity (see FT_SetDataCharacteristics)

FT_PARITY_NONE = 0
FT_PARITY_ODD = 1
FT_PARITY_EVEN = 2
FT_PARITY_MARK = 3
FT_PARITY_SPACE = 4

Flow Control (see FT_SetFlowControl)

FT_FLOW_NONE = 0x0000
FT_FLOW_RTS_CTS = 0x0100
FT_FLOW_DTR_DSR = 0x0200
FT_FLOW_XON_XOFF = 0x0400

Purge RX and TX Buffers (see FT_Purge)

FT_PURGE_RX = 1
FT_PURGE_TX = 2

Notification Events (see FT_SetEventNotification)

FT_EVENT_RXCHAR = 1
FT_EVENT_MODEM_STATUS = 2
FT_EVENT_LINE_STATUS = 4

Modem Status (see

FT_GetModemStatus)

CTS = 0x10
DSR = 0x20
RI = 0x40
DCD = 0x80

Line Status (see

FT_GetModemStatus)

OE = 0x02
PE = 0x04
FE = 0x08
BI = 0x10

Bit Modes (see FT_SetBitMode)

FT_BITMODE_RESET = 0x00
FT_BITMODE_ASYNC_BITBANG = 0x01
FT_BITMODE_MPSSE = 0x02
FT_BITMODE_SYNC_BITBANG = 0x04
FT_BITMODE_MCU_HOST = 0x08
FT_BITMODE_FAST_SERIAL = 0x10
FT_BITMODE_CBUS_BITBANG = 0x20
FT_BITMODE_SYNC_FIFO = 0x40

FT232R CBUS EEPROM OPTIONS - Ignored for FT245R (see FT_EE_Program and FT_EE_Read)

FT_232R_CBUS_TXDEN = 0x00

```
FT_232R_CBUS_PWRON = 0x01
FT_232R_CBUS_RXLED = 0x02
FT_232R_CBUS_TXLED = 0x03
FT_232R_CBUS_TXRXLED = 0x04
FT_232R_CBUS_SLEEP = 0x05
FT_232R_CBUS_CLK48 = 0x06
FT_232R_CBUS_CLK24 = 0x07
FT_232R_CBUS_CLK12 = 0x08
FT_232R_CBUS_CLK6 = 0x09
FT_232R_CBUS_IOMODE = 0x0A
FT_232R_CBUS_BITBANG_WR = 0x0B
FT_232R_CBUS_BITBANG_RD = 0x0C
```

FT232H CBUS EEPROM OPTIONS (see FT_EE_Program and FT_EE_Read)

```
FT_232H_CBUS_TRISTATE = 0x00
FT_232H_CBUS_RXLED = 0x01
FT_232H_CBUS_TXLED = 0x02
FT_232H_CBUS_TXRXLED = 0x03
FT_232H_CBUS_PWREN = 0x04
FT_232H_CBUS_SLEEP = 0x05
FT_232H_CBUS_DRIVE_0 = 0x06
FT_232H_CBUS_DRIVE_1 = 0x07
FT_232H_CBUS_IOMODE = 0x08
FT_232H_CBUS_TXDEN = 0x09
FT_232H_CBUS_CLK30 = 0x0A
FT_232H_CBUS_CLK15 = 0x0B
FT_232H_CBUS_CLK7_5 = 0x0C
```

FT_DEVICE_LIST_INFO_NODE (see FT_GetDeviceInfoList and FT_GetDeviceInfoDetail)

```
typedef struct _ft_device_list_info_node {
    DWORD Flags;
    DWORD Type;
    DWORD ID;
    DWORD LocId;
    char SerialNumber[16];
    char Description[64];
    FT_HANDLE ftHandle;
} FT_DEVICE_LIST_INFO_NODE;
```

FT_FLAGS (see [FT_DEVICE_LIST_INFO_NODE](#))

```
FT_FLAGS_OPENED = 0x00000001
```

FT_PROGRAM_DATA_STRUCTURE

```
typedef struct ft_program_data {
    DWORD Signature1;           // Header - must be 0x00000000
    DWORD Signature2;           // Header - must be 0xffffffff
    DWORD Version;              // Header - FT_PROGRAM_DATA version
                                // 0 = original (FT232B)
                                // 1 = FT2232 extensions
                                // 2 = FT232R extensions
                                // 3 = FT2232H extensions
                                // 4 = FT4232H extensions
                                // 5 = FT232H extensions
    WORD VendorId;              // 0x0403
    WORD ProductId;             // 0x6001
    char *Manufacturer;         // "FTDI"
    char *ManufacturerId;       // "FT"
    char *Description;          // "USB HS Serial Converter"
    char *SerialNumber;         // "FT000001" if fixed, or NULL
    WORD MaxPower;              // 0 <= MaxPower <= 500
}
```

```
WORD PnP;                // 0 = disabled, 1 = enabled
WORD SelfPowered;        // 0 = bus powered, 1 = self powered
WORD RemoteWakeup;       // 0 = not capable, 1 = capable
//
// Rev4 (FT232B) extensions
//
UCHAR Rev4;              // non-zero if Rev4 chip, zero otherwise
UCHAR IsoIn;             // non-zero if in endpoint is isochronous
UCHAR IsoOut;            // non-zero if out endpoint is isochronous
UCHAR PullDownEnable;    // non-zero if pull down enabled
UCHAR SerNumEnable;      // non-zero if serial number to be used
UCHAR USBVersionEnable;  // non-zero if chip uses USBVersion
WORD USBVersion;         // BCD (0x0200 => USB2)
//
// Rev 5 (FT2232) extensions
//
UCHAR Rev5;              // non-zero if Rev5 chip, zero otherwise
UCHAR IsoInA;            // non-zero if in endpoint is isochronous
UCHAR IsoInB;            // non-zero if in endpoint is isochronous
UCHAR IsoOutA;           // non-zero if out endpoint is isochronous
UCHAR IsoOutB;           // non-zero if out endpoint is isochronous
UCHAR PullDownEnable5;   // non-zero if pull down enabled
UCHAR SerNumEnable5;     // non-zero if serial number to be used
UCHAR USBVersionEnable5; // non-zero if chip uses USBVersion
WORD USBVersion5;        // BCD (0x0200 => USB2)
UCHAR AIsHighCurrent;    // non-zero if interface is high current
UCHAR BIsHighCurrent;    // non-zero if interface is high current
UCHAR IFAIsFifo;         // non-zero if interface is 245 FIFO
UCHAR IFAIsFifoTar;      // non-zero if interface is 245 FIFO CPU target
UCHAR IFAIsFastSer;      // non-zero if interface is Fast serial
UCHAR AIsVCP;            // non-zero if interface is to use VCP drivers
UCHAR IFBIsFifo;         // non-zero if interface is 245 FIFO
UCHAR IFBIsFifoTar;      // non-zero if interface is 245 FIFO CPU target
UCHAR IFBIsFastSer;      // non-zero if interface is Fast serial
UCHAR BIsVCP;            // non-zero if interface is to use VCP drivers
//
// Rev 6 (FT232R) extensions
//
UCHAR UseExtOsc;          // Use External Oscillator
UCHAR HighDriveIOs;       // High Drive I/Os
UCHAR EndpointSize;       // Endpoint size
UCHAR PullDownEnableR;    // non-zero if pull down enabled
UCHAR SerNumEnableR;      // non-zero if serial number to be used
UCHAR InvertTXD;          // non-zero if invert TXD
UCHAR InvertRXD;          // non-zero if invert RXD
UCHAR InvertRTS;          // non-zero if invert RTS
UCHAR InvertCTS;          // non-zero if invert CTS
UCHAR InvertDTR;          // non-zero if invert DTR
UCHAR InvertDSR;          // non-zero if invert DSR
UCHAR InvertDCD;          // non-zero if invert DCD
UCHAR InvertRI;           // non-zero if invert RI
UCHAR Cbus0;              // Cbus Mux control
UCHAR Cbus1;              // Cbus Mux control
UCHAR Cbus2;              // Cbus Mux control
UCHAR Cbus3;              // Cbus Mux control
UCHAR Cbus4;              // Cbus Mux control
UCHAR RIsD2XX;            // non-zero if using D2XX driver
//
// Rev 7 (FT2232H) Extensions
//
UCHAR PullDownEnable7;    // non-zero if pull down enabled
UCHAR SerNumEnable7;      // non-zero if serial number to be used
UCHAR ALSlowSlew;         // non-zero if AL pins have slow slew
```

```
UCHAR ALSchmittInput;    // non-zero if AL pins are Schmitt input
UCHAR ALDriveCurrent;    // valid values are 4mA, 8mA, 12mA, 16mA
UCHAR AHSlew;            // non-zero if AH pins have slow slew
UCHAR AHSchmittInput;    // non-zero if AH pins are Schmitt input
UCHAR AHDriveCurrent;    // valid values are 4mA, 8mA, 12mA, 16mA
UCHAR BLSlew;            // non-zero if BL pins have slow slew
UCHAR BLSchmittInput;    // non-zero if BL pins are Schmitt input
UCHAR BLDriveCurrent;    // valid values are 4mA, 8mA, 12mA, 16mA
UCHAR BHSlew;            // non-zero if BH pins have slow slew
UCHAR BHSchmittInput;    // non-zero if BH pins are Schmitt input
UCHAR BHDriveCurrent;    // valid values are 4mA, 8mA, 12mA, 16mA
UCHAR IFAIsFifo7;        // non-zero if interface is 245 FIFO
UCHAR IFAIsFifoTar7;     // non-zero if interface is 245 FIFO CPU target
UCHAR IFAIsFastSer7;     // non-zero if interface is Fast serial
UCHAR AIsVCP7;           // non-zero if interface is to use VCP drivers
UCHAR IFBIsFifo7;        // non-zero if interface is 245 FIFO
UCHAR IFBIsFifoTar7;     // non-zero if interface is 245 FIFO CPU target
UCHAR IFBIsFastSer7;     // non-zero if interface is Fast serial
UCHAR BIsVCP7;           // non-zero if interface is to use VCP drivers
UCHAR PowerSaveEnable;   // non-zero if using BCBUS7 to save power for self-powered
```

designs

```
//
// Rev 8 (FT4232H) Extensions
//
UCHAR PullDownEnable8;   // non-zero if pull down enabled
UCHAR SerNumEnable8;     // non-zero if serial number to be used
UCHAR ASlew;             // non-zero if AL pins have slow slew
UCHAR ASchmittInput;     // non-zero if AL pins are Schmitt input
UCHAR ADriveCurrent;     // valid values are 4mA, 8mA, 12mA, 16mA
UCHAR BSlew;             // non-zero if AH pins have slow slew
UCHAR BSchmittInput;     // non-zero if AH pins are Schmitt input
UCHAR BDriveCurrent;     // valid values are 4mA, 8mA, 12mA, 16mA
UCHAR CSlew;             // non-zero if BL pins have slow slew
UCHAR CSchmittInput;     // non-zero if BL pins are Schmitt input
UCHAR CDriveCurrent;     // valid values are 4mA, 8mA, 12mA, 16mA
UCHAR DSlew;             // non-zero if BH pins have slow slew
UCHAR DSchmittInput;     // non-zero if BH pins are Schmitt input
UCHAR DDriveCurrent;     // valid values are 4mA, 8mA, 12mA, 16mA
UCHAR ARIIsTXDEN;        // non-zero if port A uses RI as RS485 TXDEN
UCHAR BRIIsTXDEN;        // non-zero if port B uses RI as RS485 TXDEN
UCHAR CRIIsTXDEN;        // non-zero if port C uses RI as RS485 TXDEN
UCHAR DRIIsTXDEN;        // non-zero if port D uses RI as RS485 TXDEN
UCHAR AIsVCP8;           // non-zero if interface is to use VCP drivers
UCHAR BIsVCP8;           // non-zero if interface is to use VCP drivers
UCHAR CIsVCP8;           // non-zero if interface is to use VCP drivers
UCHAR DIsVCP8;           // non-zero if interface is to use VCP drivers
//
// Rev 9 (FT232H) Extensions
//
UCHAR PullDownEnableH;   // non-zero if pull down enabled
UCHAR SerNumEnableH;     // non-zero if serial number to be used
UCHAR ACSlew;            // non-zero if AC pins have slow slew
UCHAR ACSchmittInputH;   // non-zero if AC pins are Schmitt input
UCHAR ACDriveCurrentH;   // valid values are 4mA, 8mA, 12mA, 16mA
UCHAR ADSlew;            // non-zero if AD pins have slow slew
UCHAR ADSchmittInputH;   // non-zero if AD pins are Schmitt input
UCHAR ADDriveCurrentH;   // valid values are 4mA, 8mA, 12mA, 16mA
UCHAR Cbus0H;            // Cbus Mux control
UCHAR Cbus1H;            // Cbus Mux control
UCHAR Cbus2H;            // Cbus Mux control
UCHAR Cbus3H;            // Cbus Mux control
UCHAR Cbus4H;            // Cbus Mux control
UCHAR Cbus5H;            // Cbus Mux control
```



```

    UCHAR Cbus6H;           // Cbus Mux control
    UCHAR Cbus7H;           // Cbus Mux control
    UCHAR Cbus8H;           // Cbus Mux control
    UCHAR Cbus9H;           // Cbus Mux control
    UCHAR IsFifoH;          // non-zero if interface is 245 FIFO
    UCHAR IsFifoTarH;       // non-zero if interface is 245 FIFO CPU target
    UCHAR IsFastSerH;       // non-zero if interface is Fast serial
    UCHAR IsFT1248H;        // non-zero if interface is FT1248
    UCHAR FT1248CpolH;      // FT1248 clock polarity - clock idle high (1) or clock idle low (0)
    UCHAR FT1248LsbH;       // FT1248 data is LSB (1) or MSB (0)
    UCHAR FT1248FlowControlH; // FT1248 flow control enable
    UCHAR IsVCPH;           // non-zero if interface is to use VCP drivers
    UCHAR PowerSaveEnableH; // non-zero if using ACBUS7 to save power for self-powered

```

designs

```

} FT_PROGRAM_DATA, *PFT_PROGRAM_DATA;

```

Win32

```

OPEN_EXISTING = 3
FILE_ATTRIBUTE_NORMAL = 0x00000080
FILE_FLAG_OVERLAPPED = 0x40000000
GENERIC_READ = 0x80000000
GENERIC_WRITE = 0x40000000

```

```

OVERLAPPED structure
typedef struct _OVERLAPPED {
    ULONG_PTR Internal;
    ULONG_PTR InternalHigh;
    union {
        struct {
            DWORD Offset;
            DWORD OffsetHigh;
        };
        PVOID Pointer;
    };
    HANDLE hEvent;
} OVERLAPPED, *LPOVERLAPPED;

```

```

CLRDRTR = 6 - Clear the DTR signal
CLRRTS = 4 - Clear the RTS signal
SETDRTR = 5 - Set the DTR signal
SETRTS = 3 - Set the RTS signal
SETBREAK = 8 - Set the BREAK condition
CLRBREAK = 9 - Clear the BREAK condition

```

```

MS_CTS_ON = 0x0010 - Clear To Send (CTS) is on
MS_DSR_ON = 0x0020 - Data Set Ready (DSR) is on
MS_RING_ON = 0x0040 - Ring Indicator (RI) is on
MS_RLSD_ON = 0x0080 - Receive Line Signal Detect (RLSD) is on

```

```

FTDCB structure
typedef struct _FTDCB {
    DWORD DCBlength; // sizeof(FTDCB)
    DWORD BaudRate; // Baud rate at which running
    DWORD fBinary: 1; // Binary Mode (skip EOF check)
    DWORD fParity: 1; // Enable parity checking

```

```
DWORD fOutxCtsFlow:1; // CTS handshaking on output
DWORD fOutxDsrFlow:1; // DSR handshaking on output
DWORD fDtrControl:2; // DTR Flow control
DWORD fDsrSensitivity:1; // DSR Sensitivity
DWORD fTXContinueOnXoff: 1; // Continue TX when Xoff sent
DWORD fOutX: 1; // Enable output X-ON/X-OFF
DWORD fInX: 1; // Enable input X-ON/X-OFF
DWORD fErrorChar: 1; // Enable Err Replacement
DWORD fNull: 1; // Enable Null stripping
DWORD fRtsControl:2; // Rts Flow control
DWORD fAbortOnError:1; // Abort all reads and writes on Error
DWORD fDummy2:17; // Reserved
WORD wReserved; // Not currently used
WORD XonLim; // Transmit X-ON threshold
WORD XoffLim; // Transmit X-OFF threshold
BYTE ByteSize; // Number of bits/byte, 7-8
BYTE Parity; // 0-4=None,Odd,Even,Mark,Space
BYTE StopBits; // 0,2 = 1, 2
char XonChar; // Tx and Rx X-ON character
char XoffChar; // Tx and Rx X-OFF character
char ErrorChar; // Error replacement char
char EofChar; // End of Input character
char EvtChar; // Received Event character
WORD wReserved1; // Fill
} FTDCB, *LPFTDCB;

FTTIMEOUTS structure
typedef struct _FTTIMEOUTS {
    DWORD ReadIntervalTimeout; // Maximum time between read chars
    DWORD ReadTotalTimeoutMultiplier; // Multiplier of characters
    DWORD ReadTotalTimeoutConstant; // Constant in milliseconds
    DWORD WriteTotalTimeoutMultiplier; // Multiplier of characters
    DWORD WriteTotalTimeoutConstant; // Constant in milliseconds
} FTTIMEOUTS, *LPFTTIMEOUTS;

EV_BREAK = 0x0040 - BREAK condition detected
EV_CTS = 0x0008 - Change in Clear To Send (CTS)
EV_DSR = 0x0010 - Change in Data Set Ready (DSR)
EV_ERR = 0x0080 - Error in line status
EV_RING = 0x0100 - Change in Ring Indicator (RI)
EV_RLSD = 0x0020 - Change in Receive Line Signal Detect (RLSD)
EV_RXCHAR = 0x0001 - Character received
EV_RXFLAG = 0x0002 - Event character received
EV_TXEMPTY = 0x0004 - Transmitter empty

PURGE_TXABORT = 0x0001 - Terminate outstanding overlapped writes
PURGE_RXABORT = 0x0002 - Terminate outstanding overlapped reads
PURGE_TXCLEAR = 0x0004 - Clear the transmit buffer
PURGE_RXCLEAR = 0x0008 - Clear the receive buffer
```

```
FTCOMSTAT structure
typedef struct _FTCOMSTAT {
    DWORD fCtsHold : 1;
    DWORD fDsrHold : 1;
    DWORD fRlsdHold : 1;
    DWORD fXoffHold : 1;
    DWORD fXoffSent : 1;
    DWORD fEof : 1;
    DWORD fTxim : 1;
    DWORD fReserved : 25;
```

```
DWORD cbInQue;  
    DWORD cbOutQue;  
} FTCOMSTAT, *LPFTCOMSTAT;
```

9 Appendix B - Revision History

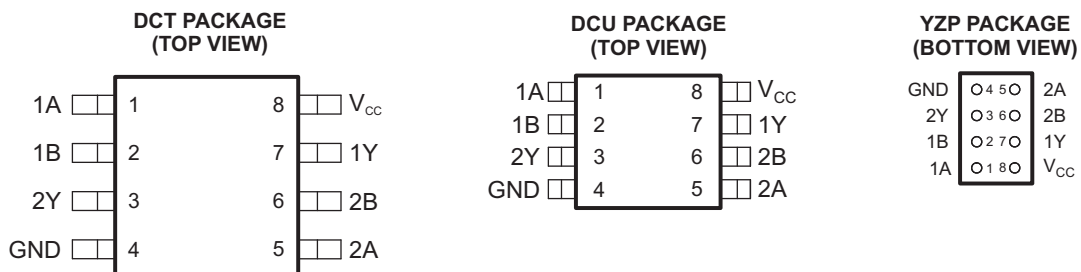
Revision History

Revision History

Version 1.00	Initial release in new format. Includes all functions in CDM driver 2.04.06	August, 2008
Version 1.01	Includes FT4232H and FT2232H. Updated addresses	January, 2009
Version 1.02	Page 65 – removed FT232R and FT245R reference from MCU host emulation and Fast Opto modes	January, 2010
Version 1.03	Corrected section 3.32 (FT_Purge) Updated Contact details	8th September 2010
Version 1.04	Added 245 Synchronous FIFO mode code in section 5.3	28th October 2010
Version 1.1	Corrected previous editing errors to the document by re-adding FT4232H and FT2232H extensions	4th November 2010
Version 1.2	Added references to FT232H including EEPROM format. Numerous formatting fixes. Expanded definitions in appendix to reflect updates in CDM 2.08.14 header file.	25 th April 2011

FEATURES

- Available in Texas Instruments NanoFree™ Package
- Supports 5-V V_{CC} Operation
- Inputs Accept Voltages to 5.5 V
- Max t_{pd} of 5.3 ns at 3.3 V
- Low Power Consumption, 10- μ A Max I_{CC}
- ± 24 -mA Output Drive at 3.3 V
- Typical V_{OLP} (Output Ground Bounce) < 0.8 V at $V_{CC} = 3.3$ V, $T_A = 25^\circ\text{C}$
- Typical V_{OHV} (Output V_{OH} Undershoot) > 2 V at $V_{CC} = 3.3$ V, $T_A = 25^\circ\text{C}$
- I_{off} Supports Partial-Power-Down Mode Operation
- Latch-Up Performance Exceeds 100 mA Per JESD 78, Class II
- ESD Protection Exceeds JESD 22
 - 2000-V Human-Body Model (A114-A)
 - 200-V Machine Model (A115-A)
 - 1000-V Charged-Device Model (C101)



See mechanical drawings for dimensions.

DESCRIPTION/ORDERING INFORMATION

This dual 2-input NAND gate with Schmitt-trigger inputs is designed for 1.65-V to 5.5-V V_{CC} operation.

The SN74LVC2G132 contains two inverters and performs the Boolean function $Y = \overline{A \cdot B}$ or $Y = \overline{A} + \overline{B}$ in positive logic. The device functions as two independent inverters, but because of Schmitt action, it has different input threshold levels for positive-going (V_{T+}) and negative-going (V_{T-}) signals.

NanoFree™ package technology is a major breakthrough in IC packaging concepts, using the die as the package.

This device can be triggered from the slowest of input ramps and still give clean jitter-free output signals.

This device is fully specified for partial-power-down applications using I_{off} . The I_{off} circuitry disables the outputs, preventing damaging current backflow through the device when it is powered down.

ORDERING INFORMATION

T_A	PACKAGE ⁽¹⁾		ORDERABLE PART NUMBER	TOP-SIDE MARKING ⁽²⁾
-40°C to 85°C	NanoFree™ – WCSP (DSBGA) 0.23-mm Large Bump – YZP (Pb-free)	Reel of 3000	SN74LVC2G132YZPR	__ _D5_
		Reel of 3000	SN74LVC2G132DCTR	C3B_ _ _
	SSOP – DCT	Reel of 3000	SN74LVC2G132DCUR	C3B_
		Reel of 250	SN74LVC2G132DCUT	

(1) Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

(2) DCT: The actual top-side marking has three additional characters that designate the year, month, and assembly/test site.

DCU: The actual top-side marking has one additional character that designates the assembly/test site.

YZP: The actual top-side marking has three preceding characters to denote year, month, and sequence code, and one following character to designate the assembly/test site. Pin 1 identifier indicates solder-bump composition (1 = SnPb, • = Pb-free).



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

NanoFree is a trademark of Texas Instruments.

SN74LVC2G132

DUAL 2-INPUT NAND GATE

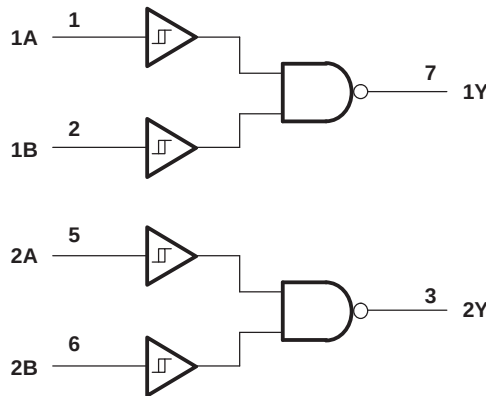
WITH SCHMITT-TRIGGER INPUTS

SCES547C—FEBRUARY 2004—REVISED JANUARY 2007

FUNCTION TABLE
(EACH GATE)

INPUTS		OUTPUT Y
A	B	
L	L	H
L	H	H
H	L	H
H	H	L

LOGIC DIAGRAM (POSITIVE LOGIC)



Absolute Maximum Ratings⁽¹⁾

over operating free-air temperature range (unless otherwise noted)

			MIN	MAX	UNIT
V _{CC}	Supply voltage range		−0.5	6.5	V
V _I	Input voltage range ⁽²⁾		−0.5	6.5	V
V _O	Voltage range applied to any output in the high-impedance or power-off state ⁽²⁾		−0.5	6.5	V
V _O	Voltage range applied to any output in the high or low state ⁽²⁾⁽³⁾		−0.5	V _{CC} + 0.5	V
I _{IK}	Input clamp current	V _I < 0	−50		mA
I _{OK}	Output clamp current	V _O < 0	−50		mA
I _O	Continuous output current		±50		mA
Continuous current through V _{CC} or GND			±100		mA
θ _{JA}	Package thermal impedance ⁽⁴⁾	DCT package	220		°C/W
		DCU package	227		
		YZP package	102		
T _{stg}	Storage temperature range		−65	150	°C

- (1) Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.
- (2) The input and output negative-voltage ratings may be exceeded if the input and output current ratings are observed.
- (3) The value of V_{CC} is provided in the recommended operating conditions table.
- (4) The package thermal impedance is calculated in accordance with JESD 51-7

Recommended Operating Conditions⁽¹⁾

			MIN	MAX	UNIT
V _{CC}	Supply voltage	Operating	1.65	5.5	V
		Data retention only	1.5		
V _I	Input voltage		0	5.5	V
V _O	Output voltage		0	V _{CC}	V
I _{OH}	High-level output current	V _{CC} = 1.65 V		−4	mA
		V _{CC} = 2.3 V		−8	
		V _{CC} = 3 V		−16	
				−24	
		V _{CC} = 4.5 V		−32	
I _{OL}	Low-level output current	V _{CC} = 1.65 V		4	mA
		V _{CC} = 2.3 V		8	
		V _{CC} = 3 V		16	
				24	
		V _{CC} = 4.5 V		32	
T _A	Operating free-air temperature		−40	85	°C

- (1) All unused inputs of the device must be held at V_{CC} or GND to ensure proper device operation. Refer to the TI application report, *Implications of Slow or Floating CMOS Inputs*, literature number SCBA004.

SN74LVC2G132

DUAL 2-INPUT NAND GATE

WITH SCHMITT-TRIGGER INPUTS

SCES547C – FEBRUARY 2004 – REVISED JANUARY 2007

Electrical Characteristics

over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS	V _{CC}	MIN	TYP ⁽¹⁾	MAX	UNIT
V _{T+} Positive-going input threshold voltage		1.65 V	0.79		1.16	V
		2.3 V	1.11		1.56	
		3 V	1.5		1.87	
		4.5 V	2.16		2.74	
		5.5 V	2.61		3.33	
V _{T-} Negative-going input threshold voltage		1.65 V	0.39		0.62	V
		2.3 V	0.58		0.87	
		3 V	0.84		1.14	
		4.5 V	1.41		1.79	
		5.5 V	1.87		2.29	
ΔV_T Hysteresis (V _{T+} – V _{T-})		1.65 V	0.37		0.62	V
		2.3 V	0.48		0.77	
		3 V	0.56		0.87	
		4.5 V	0.71		1.04	
		5.5 V	0.71		1.11	
V _{OH}	I _{OH} = –100 μ A	1.65 V to 5.5 V	V _{CC} – 0.1			V
	I _{OH} = –4 mA	1.65 V	1.2			
	I _{OH} = –8 mA	2.3 V	1.9			
	I _{OH} = –16 mA	3 V	2.4			
	I _{OH} = –24 mA		2.3			
	I _{OH} = –32 mA	4.5 V	3.8			
V _{OL}	I _{OL} = 100 μ A	1.65 V to 5.5 V	0.1			V
	I _{OL} = 4 mA	1.65 V	0.45			
	I _{OL} = 8 mA	2.3 V	0.3			
	I _{OL} = 16 mA	3 V	0.4			
	I _{OL} = 24 mA		0.55			
	I _{OL} = 32 mA	4.5 V	0.55			
I _I	A or B inputs	V _I = 5.5 V or GND	1.65 V to 5.5 V	± 1		μ A
I _{off}	V _I or V _O = 5.5 V	0	± 10			μ A
I _{CC}	V _I = V _{CC} or GND, I _O = 0	1.65 V to 5.5 V	10			μ A
ΔI_{CC}	One input at V _{CC} – 0.6 V, Other inputs at V _{CC} or GND	3 V to 5.5 V	500			μ A
C _I	V _I = V _{CC} or GND	3.3 V	3.5			pF

(1) All typical values are at V_{CC} = 3.3 V, T_A = 25°C.

Switching Characteristics

over recommended operating free-air temperature range, $C_L = 15 \text{ pF}$ (unless otherwise noted) (see [Figure 1](#))

PARAMETER	FROM (INPUT)	TO (OUTPUT)	$V_{CC} = 1.8 \text{ V}$ $\pm 0.15 \text{ V}$		$V_{CC} = 2.5 \text{ V}$ $\pm 0.2 \text{ V}$		$V_{CC} = 3.3 \text{ V}$ $\pm 0.3 \text{ V}$		$V_{CC} = 5 \text{ V}$ $\pm 0.5 \text{ V}$		UNIT
			MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX	
t_{pd}	A or B	Y	4	16	2.5	7	2	5.3	1.5	4.4	ns

Switching Characteristics

over recommended operating free-air temperature range, $C_L = 30 \text{ pF}$ or 50 pF (unless otherwise noted) (see [Figure 2](#))

PARAMETER	FROM (INPUT)	TO (OUTPUT)	$V_{CC} = 1.8 \text{ V}$ $\pm 0.15 \text{ V}$		$V_{CC} = 2.5 \text{ V}$ $\pm 0.2 \text{ V}$		$V_{CC} = 3.3 \text{ V}$ $\pm 0.3 \text{ V}$		$V_{CC} = 5 \text{ V}$ $\pm 0.5 \text{ V}$		UNIT
			MIN	MAX	MIN	MAX	MIN	MAX	MIN	MAX	
t_{pd}	A or B	Y	4	16	3	7.5	2	6	2	5	ns

Operating Characteristics

$T_A = 25^\circ\text{C}$

PARAMETER		TEST CONDITIONS	$V_{CC} = 1.8 \text{ V}$	$V_{CC} = 2.5 \text{ V}$	$V_{CC} = 3.3 \text{ V}$	$V_{CC} = 5 \text{ V}$	UNIT
			TYP	TYP	TYP	TYP	
C_{pd}	Power dissipation capacitance	$f = 10 \text{ MHz}$	17	18	18	20	pF

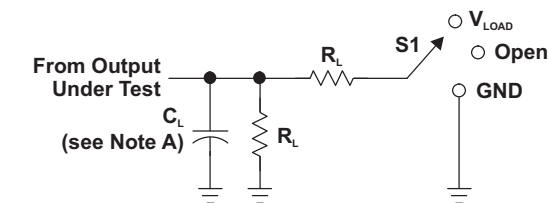
SN74LVC2G132

DUAL 2-INPUT NAND GATE

WITH SCHMITT-TRIGGER INPUTS

SCES547C—FEBRUARY 2004—REVISED JANUARY 2007

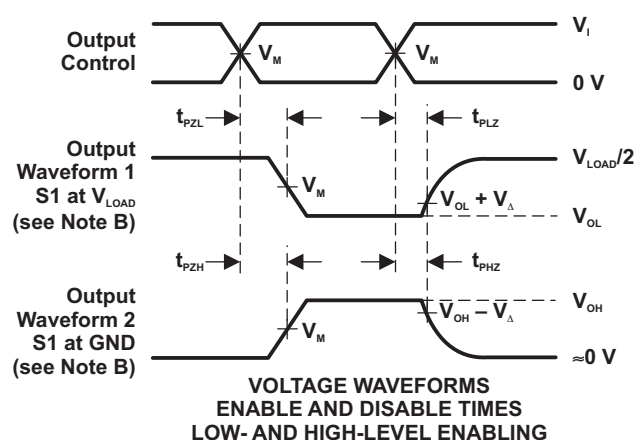
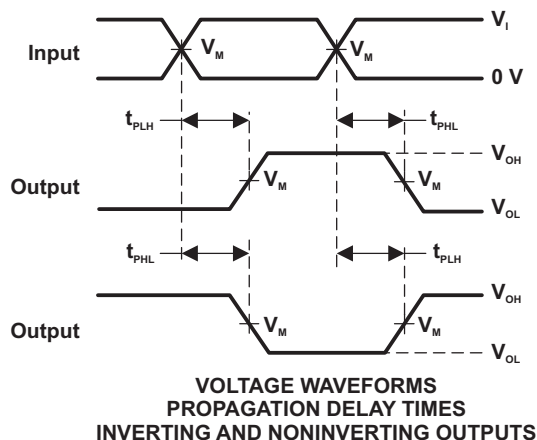
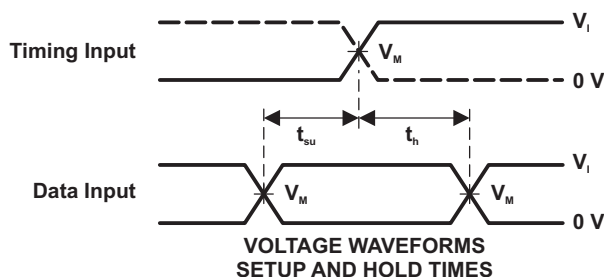
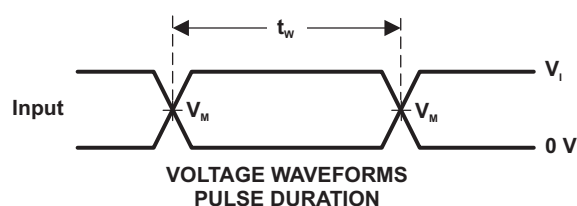
PARAMETER MEASUREMENT INFORMATION



LOAD CIRCUIT

TEST	S1
t_{PLH}/t_{PHL}	Open
t_{PLZ}/t_{PZL}	V_{LOAD}
t_{PHZ}/t_{PZH}	GND

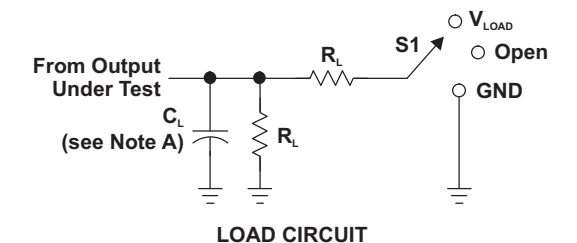
V_{CC}	INPUTS		V_M	V_{LOAD}	C_L	R_L	V_{Δ}
	V_I	t_i/t_f					
$1.8\text{ V} \pm 0.15\text{ V}$	V_{CC}	$\leq 2\text{ ns}$	$V_{CC}/2$	$2 \times V_{CC}$	15 pF	1 M Ω	0.15 V
$2.5\text{ V} \pm 0.2\text{ V}$	V_{CC}	$\leq 2\text{ ns}$	$V_{CC}/2$	$2 \times V_{CC}$	15 pF	1 M Ω	0.15 V
$3.3\text{ V} \pm 0.3\text{ V}$	3 V	$\leq 2.5\text{ ns}$	1.5 V	6 V	15 pF	1 M Ω	0.3 V
$5\text{ V} \pm 0.5\text{ V}$	V_{CC}	$\leq 2.5\text{ ns}$	$V_{CC}/2$	$2 \times V_{CC}$	15 pF	1 M Ω	0.3 V



- NOTES:
- C_L includes probe and jig capacitance.
 - Waveform 1 is for an output with internal conditions such that the output is low, except when disabled by the output control. Waveform 2 is for an output with internal conditions such that the output is high, except when disabled by the output control.
 - All input pulses are supplied by generators having the following characteristics: $PRR \leq 10\text{ MHz}$, $Z_o = 50\ \Omega$.
 - The outputs are measured one at a time, with one transition per measurement.
 - t_{PLZ} and t_{PHZ} are the same as t_{dis} .
 - t_{PZL} and t_{PZH} are the same as t_{en} .
 - t_{PLH} and t_{PHL} are the same as t_{pd} .
 - All parameters and waveforms are not applicable to all devices.

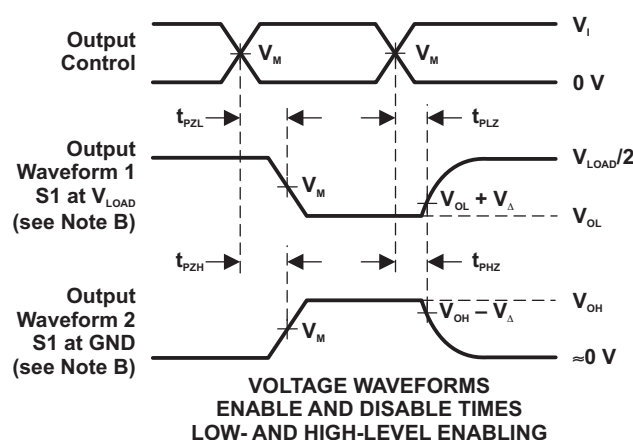
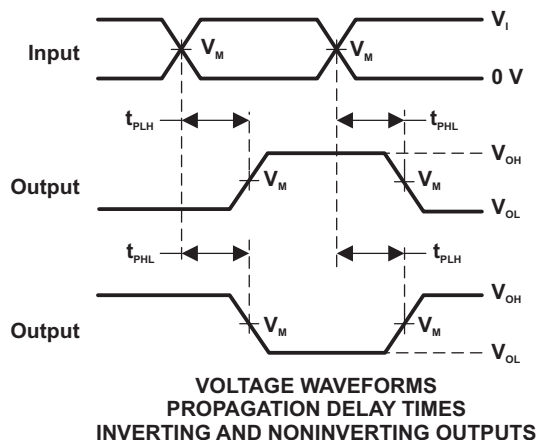
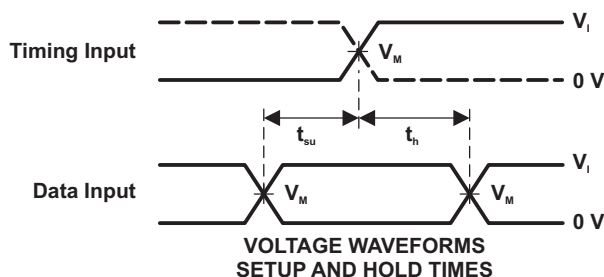
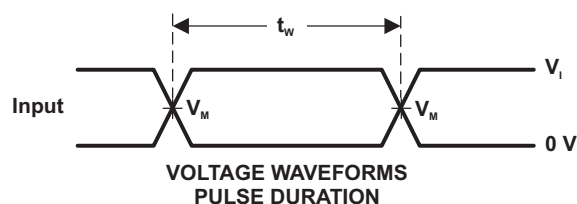
Figure 1. Load Circuit and Voltage Waveforms

PARAMETER MEASUREMENT INFORMATION



TEST	S1
t_{PLH}/t_{PHL}	Open
t_{PLZ}/t_{PZL}	V_{LOAD}
t_{PHZ}/t_{PZH}	GND

V_{CC}	INPUTS		V_M	V_{LOAD}	C_L	R_L	V_{Δ}
	V_I	t_I/t_f					
$1.8\text{ V} \pm 0.15\text{ V}$	V_{CC}	$\leq 2\text{ ns}$	$V_{CC}/2$	$2 \times V_{CC}$	30 pF	1 k Ω	0.15 V
$2.5\text{ V} \pm 0.2\text{ V}$	V_{CC}	$\leq 2\text{ ns}$	$V_{CC}/2$	$2 \times V_{CC}$	30 pF	500 Ω	0.15 V
$3.3\text{ V} \pm 0.3\text{ V}$	3 V	$\leq 2.5\text{ ns}$	1.5 V	6 V	50 pF	500 Ω	0.3 V
$5\text{ V} \pm 0.5\text{ V}$	V_{CC}	$\leq 2.5\text{ ns}$	$V_{CC}/2$	$2 \times V_{CC}$	50 pF	500 Ω	0.3 V



- NOTES:
- C_L includes probe and jig capacitance.
 - Waveform 1 is for an output with internal conditions such that the output is low, except when disabled by the output control. Waveform 2 is for an output with internal conditions such that the output is high, except when disabled by the output control.
 - All input pulses are supplied by generators having the following characteristics: $PRR \leq 10\text{ MHz}$, $Z_o = 50\text{ }\Omega$.
 - The outputs are measured one at a time, with one transition per measurement.
 - t_{PLZ} and t_{PHZ} are the same as t_{dis} .
 - t_{PZL} and t_{PZH} are the same as t_{en} .
 - t_{PLH} and t_{PHL} are the same as t_{pd} .
 - All parameters and waveforms are not applicable to all devices.

Figure 2. Load Circuit and Voltage Waveforms

PACKAGING INFORMATION

Orderable Device	Status ⁽¹⁾	Package Type	Package Drawing	Pins	Package Qty	Eco Plan ⁽²⁾	Lead/Ball Finish	MSL Peak Temp ⁽³⁾
74LVC2G132DCTRE4	ACTIVE	SM8	DCT	8	3000	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
74LVC2G132DCTRG4	ACTIVE	SM8	DCT	8	3000	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
74LVC2G132DCURE4	ACTIVE	US8	DCU	8	3000	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
74LVC2G132DCURG4	ACTIVE	US8	DCU	8	3000	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
74LVC2G132DCUTE4	ACTIVE	US8	DCU	8	250	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN74LVC2G132DCTR	ACTIVE	SM8	DCT	8	3000	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN74LVC2G132DCUR	ACTIVE	US8	DCU	8	3000	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN74LVC2G132DCUT	ACTIVE	US8	DCU	8	250	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN74LVC2G132YZPR	ACTIVE	DSBGA	YZP	8	3000	Green (RoHS & no Sb/Br)	SNAGCU	Level-1-260C-UNLIM

⁽¹⁾ The marketing status values are defined as follows:

ACTIVE: Product device recommended for new designs.

LIFEBUY: TI has announced that the device will be discontinued, and a lifetime-buy period is in effect.

NRND: Not recommended for new designs. Device is in production to support existing customers, but TI does not recommend using this part in a new design.

PREVIEW: Device has been announced but is not in production. Samples may or may not be available.

OBsolete: TI has discontinued the production of the device.

⁽²⁾ Eco Plan - The planned eco-friendly classification: Pb-Free (RoHS), Pb-Free (RoHS Exempt), or Green (RoHS & no Sb/Br) - please check <http://www.ti.com/productcontent> for the latest availability information and additional product content details.

TBD: The Pb-Free/Green conversion plan has not been defined.

Pb-Free (RoHS): TI's terms "Lead-Free" or "Pb-Free" mean semiconductor products that are compatible with the current RoHS requirements for all 6 substances, including the requirement that lead not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, TI Pb-Free products are suitable for use in specified lead-free processes.

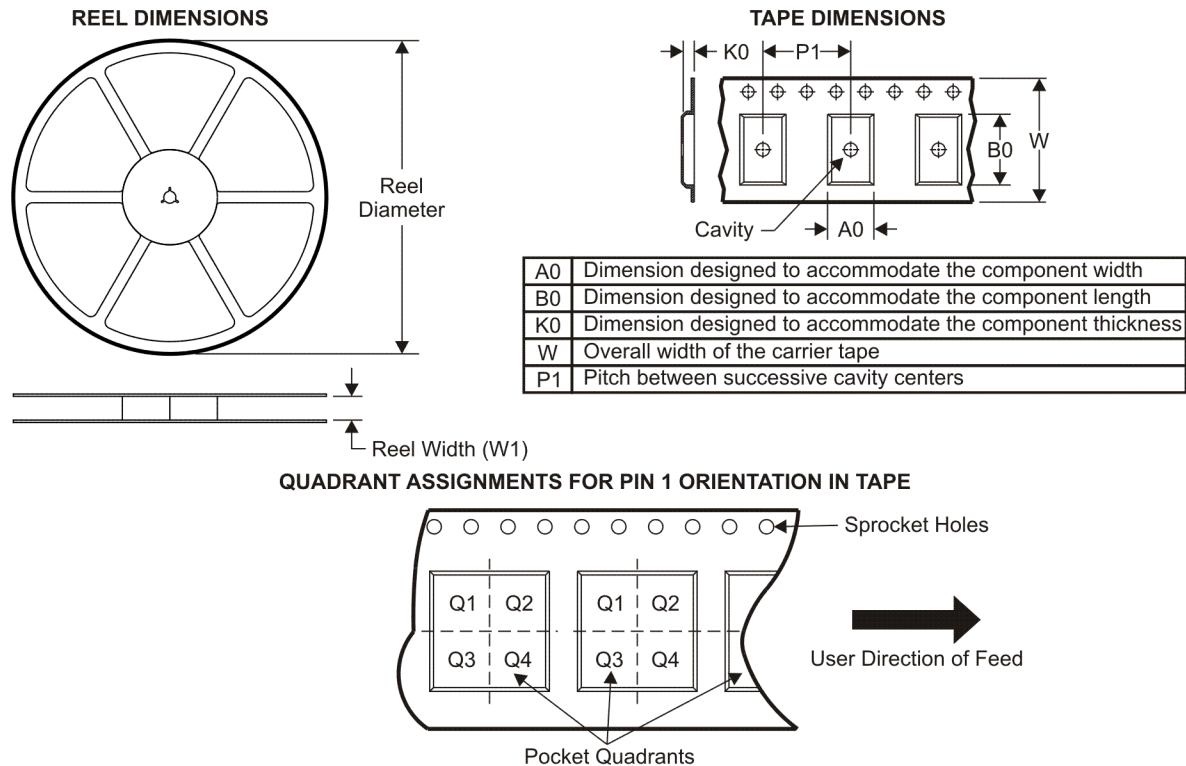
Pb-Free (RoHS Exempt): This component has a RoHS exemption for either 1) lead-based flip-chip solder bumps used between the die and package, or 2) lead-based die adhesive used between the die and leadframe. The component is otherwise considered Pb-Free (RoHS compatible) as defined above.

Green (RoHS & no Sb/Br): TI defines "Green" to mean Pb-Free (RoHS compatible), and free of Bromine (Br) and Antimony (Sb) based flame retardants (Br or Sb do not exceed 0.1% by weight in homogeneous material)

⁽³⁾ MSL, Peak Temp. -- The Moisture Sensitivity Level rating according to the JEDEC industry standard classifications, and peak solder temperature.

Important Information and Disclaimer: The information provided on this page represents TI's knowledge and belief as of the date that it is provided. TI bases its knowledge and belief on information provided by third parties, and makes no representation or warranty as to the accuracy of such information. Efforts are underway to better integrate information from third parties. TI has taken and continues to take reasonable steps to provide representative and accurate information but may not have conducted destructive testing or chemical analysis on incoming materials and chemicals. TI and TI suppliers consider certain information to be proprietary, and thus CAS numbers and other limited information may not be available for release.

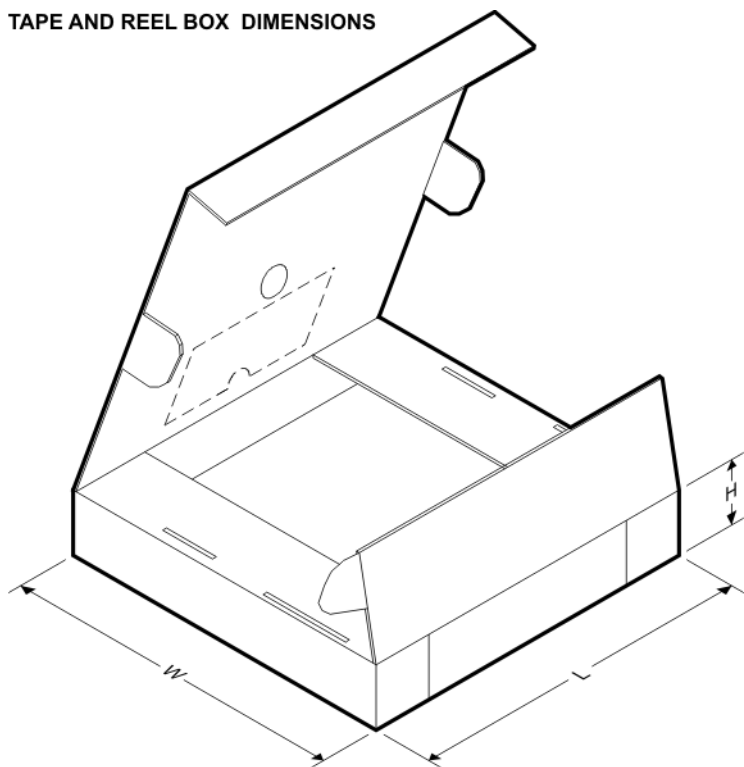
In no event shall TI's liability arising out of such information exceed the total purchase price of the TI part(s) at issue in this document sold by TI to Customer on an annual basis.

TAPE AND REEL INFORMATION


*All dimensions are nominal

Device	Package Type	Package Drawing	Pins	SPQ	Reel Diameter (mm)	Reel Width W1 (mm)	A0 (mm)	B0 (mm)	K0 (mm)	P1 (mm)	W (mm)	Pin1 Quadrant
SN74LVC2G132YZPR	DSBGA	YZP	8	3000	180.0	8.4	1.02	2.02	0.63	4.0	8.0	Q1

TAPE AND REEL BOX DIMENSIONS

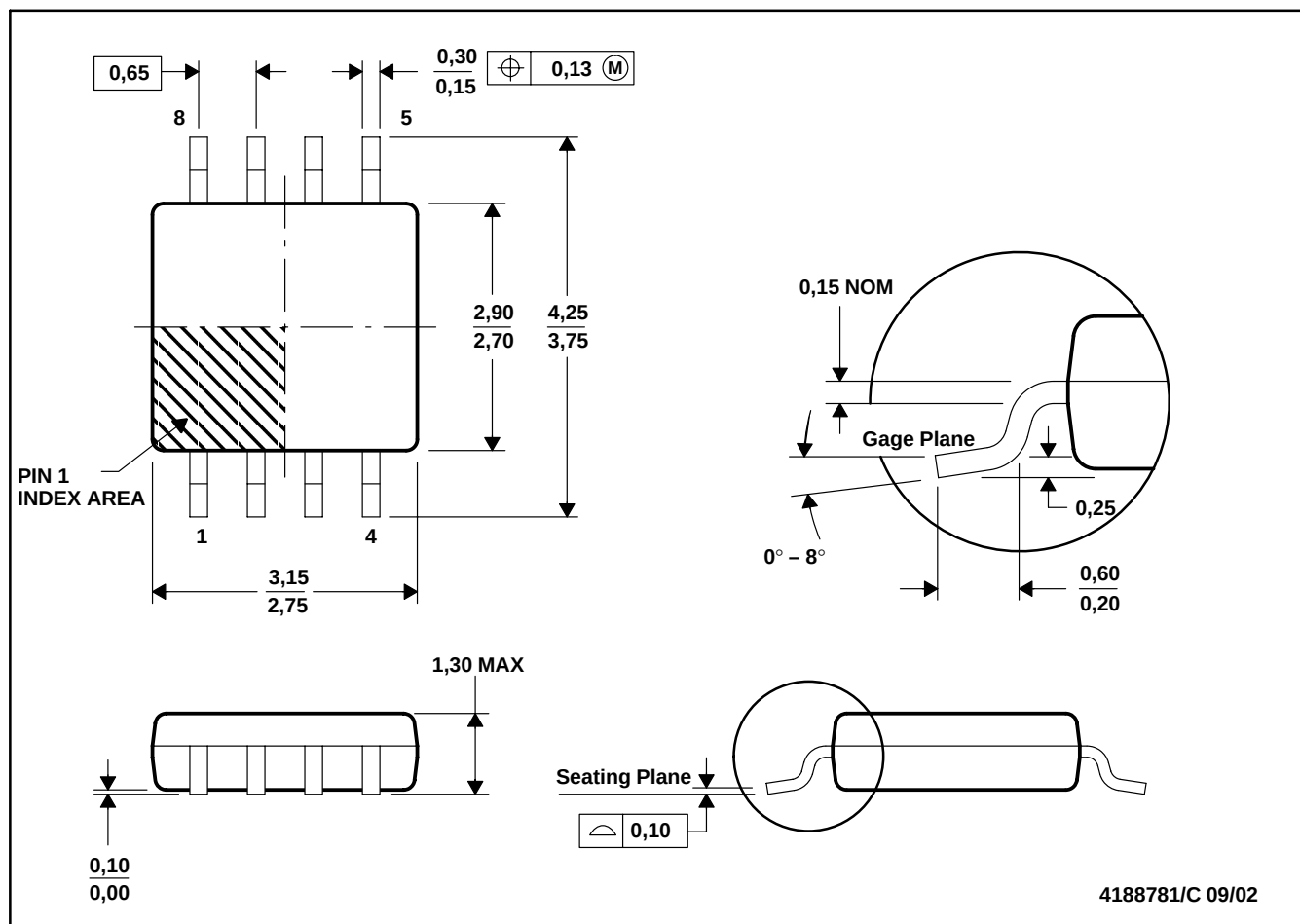


*All dimensions are nominal

Device	Package Type	Package Drawing	Pins	SPQ	Length (mm)	Width (mm)	Height (mm)
SN74LVC2G132YZPR	DSBGA	YZP	8	3000	220.0	220.0	34.0

DCT (R-PDSO-G8)

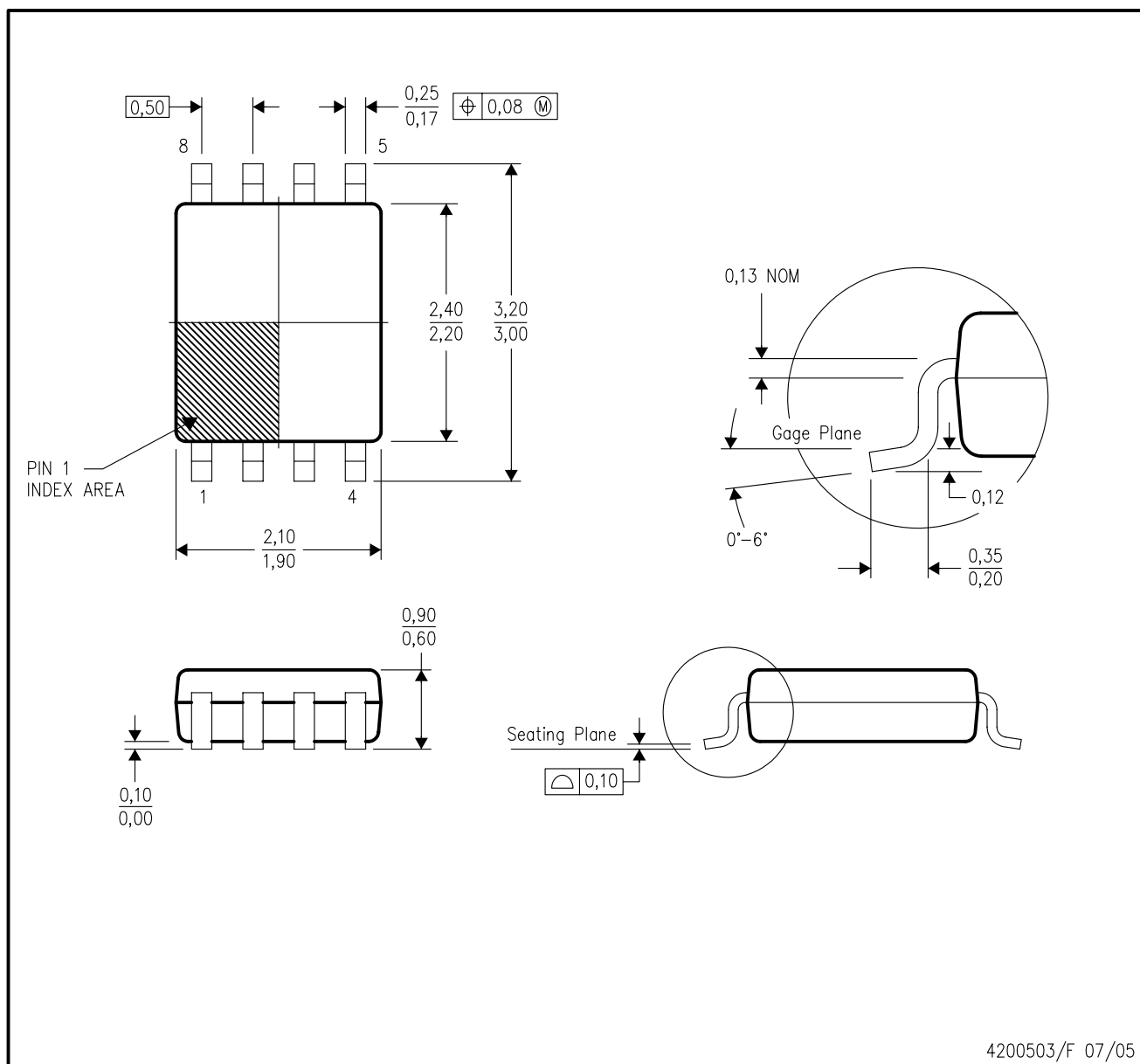
PLASTIC SMALL-OUTLINE PACKAGE



- NOTES:
- All linear dimensions are in millimeters.
 - This drawing is subject to change without notice.
 - Body dimensions do not include mold flash or protrusion.
 - Falls within JEDEC MO-187 variation DA.

DCU (R-PDSO-G8)

PLASTIC SMALL-OUTLINE PACKAGE (DIE DOWN)

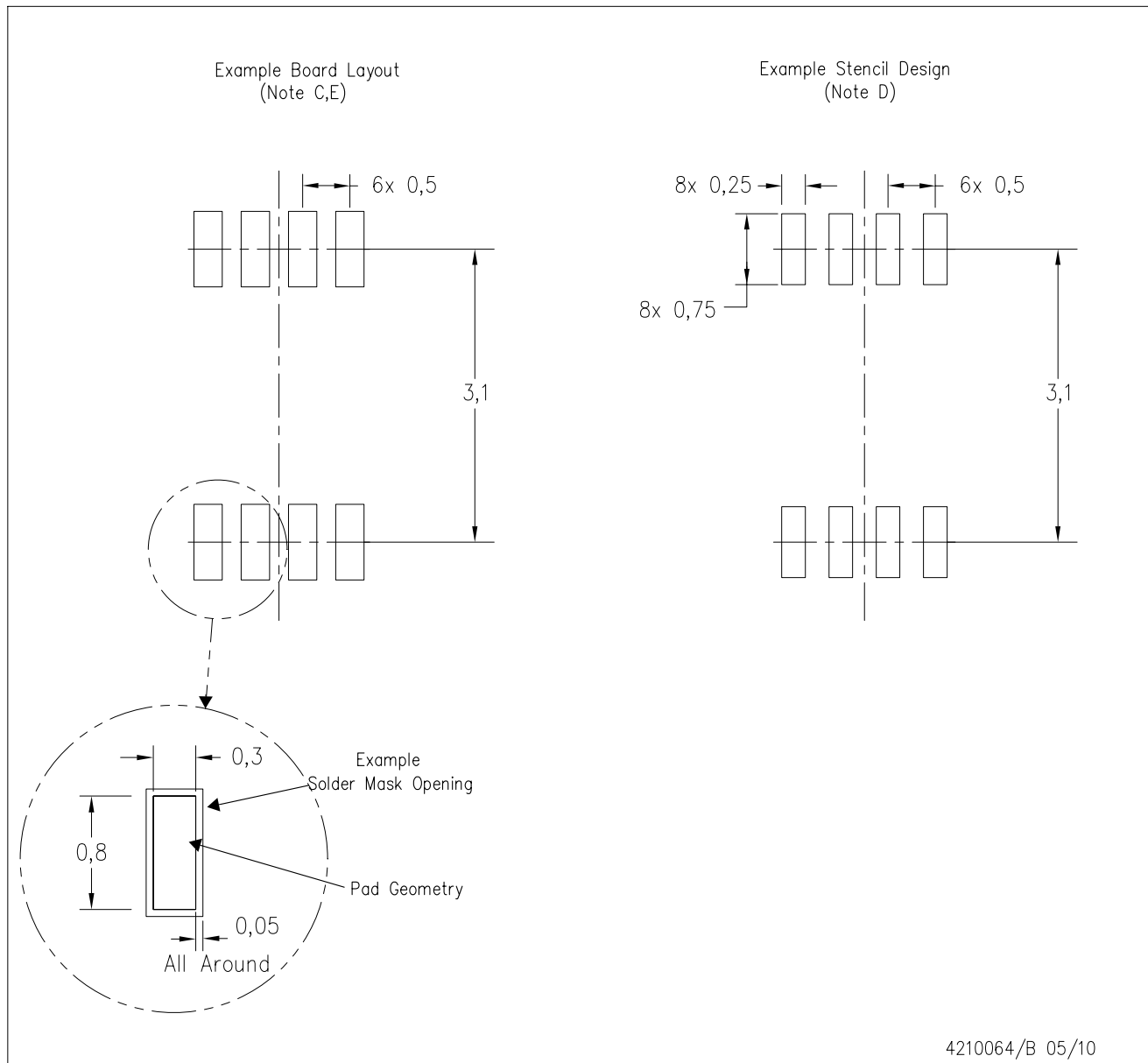


NOTES:

- All linear dimensions are in millimeters.
- This drawing is subject to change without notice.
- Body dimensions do not include mold flash or protrusion. Mold flash and protrusion shall not exceed 0.15 per side.
- Falls within JEDEC MO-187 variation CA.

DCU (S-PDSO-G8)

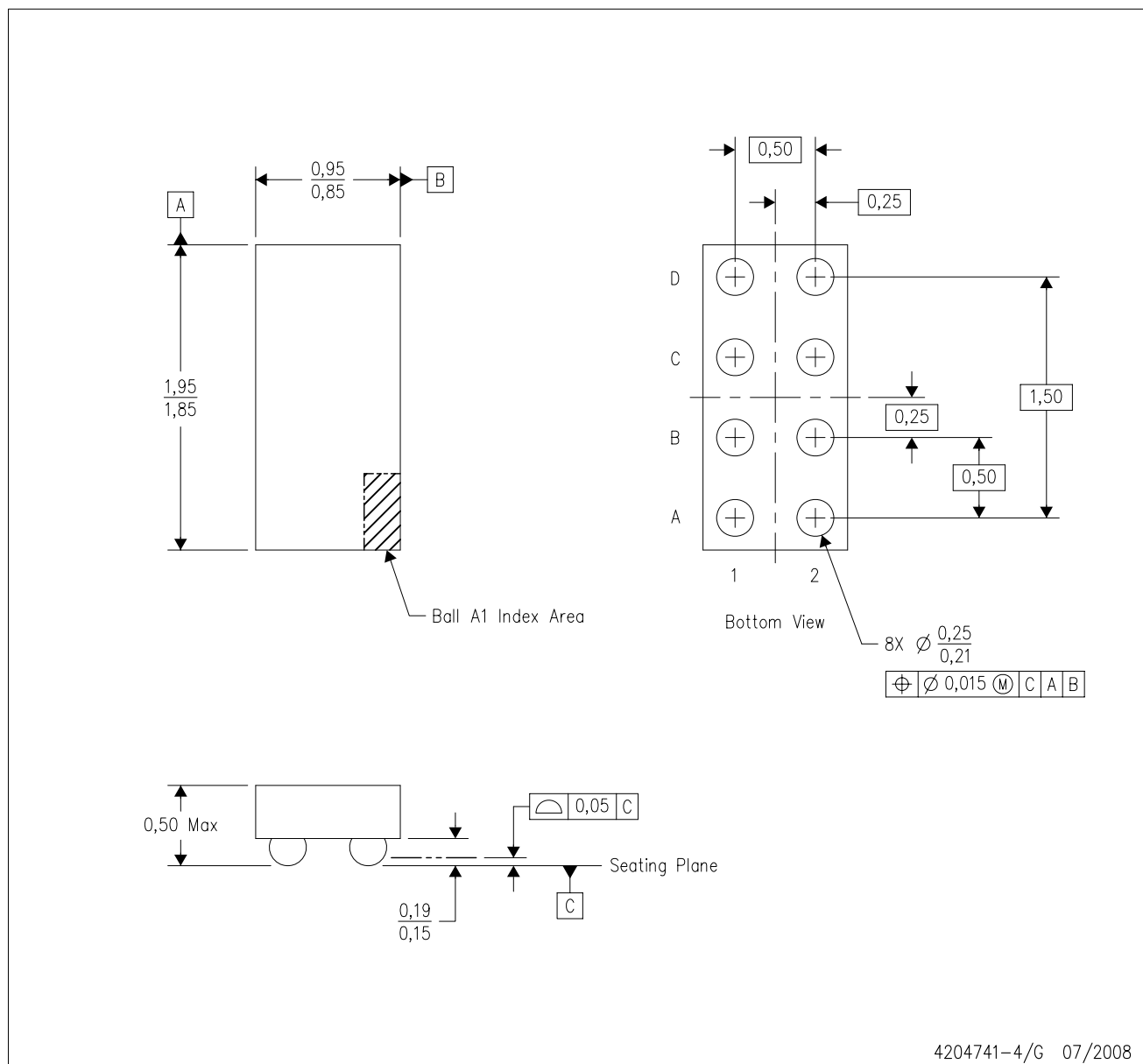
PLASTIC SMALL OUTLINE PACKAGE (DIE DOWN)



- NOTES:
- A. All linear dimensions are in millimeters.
 - B. This drawing is subject to change without notice.
 - C. Publication IPC-7351 is recommended for alternate designs.
 - D. Laser cutting apertures with trapezoidal walls and also rounding corners will offer better paste release. Customers should contact their board assembly site for stencil design recommendations. Refer to IPC-7525.
 - E. Customers should contact their board fabrication site for solder mask tolerances between and around signal pads.

YZP (R-XBGA-N8)

DIE-SIZE BALL GRID ARRAY



- NOTES:
- A. All linear dimensions are in millimeters. Dimensioning and tolerancing per ASME Y14.5M-1994.
 - B. This drawing is subject to change without notice.
 - C. NanoFree™ package configuration.
 - D. This package is lead-free. Refer to the 8 YEP package (drawing 4204725) for tin-lead (SnPb).

NanoFree is a trademark of Texas Instruments.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products

Audio	www.ti.com/audio
Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
RF/IF and ZigBee® Solutions	www.ti.com/lprf

Applications

Communications and Telecom	www.ti.com/communications
Computers and Peripherals	www.ti.com/computers
Consumer Electronics	www.ti.com/consumer-apps
Energy and Lighting	www.ti.com/energy
Industrial	www.ti.com/industrial
Medical	www.ti.com/medical
Security	www.ti.com/security
Space, Avionics and Defense	www.ti.com/space-avionics-defense
Transportation and Automotive	www.ti.com/automotive
Video and Imaging	www.ti.com/video
Wireless	www.ti.com/wireless-apps

TI E2E Community Home Page

e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2011, Texas Instruments Incorporated

Features

- 7.5 ns pin-to-pin logic delays on all pins
- f_{CNT} to 125 MHz
- 72 macrocells with 1,600 usable gates
- Up to 72 user I/O pins
- 5V in-system programmable
 - Endurance of 10,000 program/erase cycles
 - Program/erase over full commercial voltage and temperature range
- Enhanced pin-locking architecture
- Flexible 36V18 Function Block
 - 90 product terms drive any or all of 18 macrocells within Function Block
 - Global and product term clocks, output enables, set and reset signals
- Extensive IEEE Std 1149.1 boundary-scan (JTAG) support
- Programmable power reduction mode in each macrocell
- Slew rate control on individual outputs
- User programmable ground pin capability
- Extended pattern security features for design protection
- High-drive 24 mA outputs
- 3.3V or 5V I/O capability
- Advanced CMOS 5V FastFLASH™ technology
- Supports parallel programming of more than one XC9500 concurrently
- Available in 44-pin PLCC, 84-pin PLCC, 100-pin PQFP, and 100-pin TQFP packages

Description

The XC9572 is a high-performance CPLD providing advanced in-system programming and test capabilities for general purpose logic integration. It is comprised of eight 36V18 Function Blocks, providing 1,600 usable gates with propagation delays of 7.5 ns. See Figure 2 for the architecture overview.

Power Management

Power dissipation can be reduced in the XC9572 by configuring macrocells to standard or low-power modes of operation. Unused macrocells are turned off to minimize power dissipation.

Operating current for each design can be approximated for specific operating conditions using the following equation:

$$I_{CC} \text{ (mA)} = MC_{HP} (1.7) + MC_{LP} (0.9) + MC (0.006 \text{ mA/MHz}) f$$

Where:

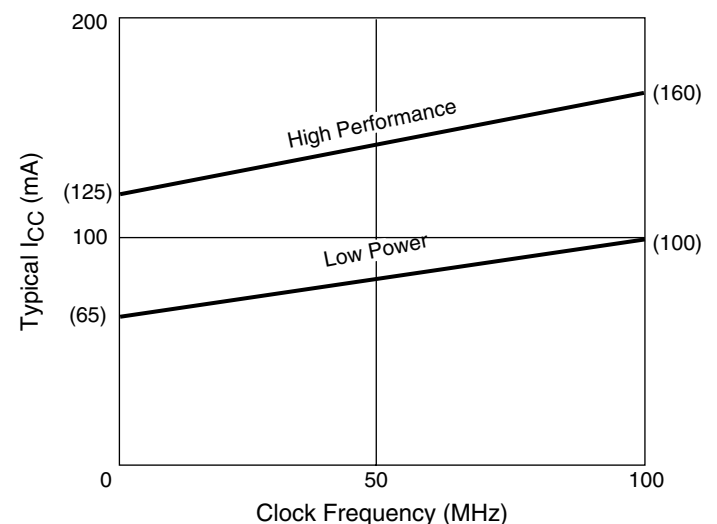
MC_{HP} = Macrocells in high-performance mode

MC_{LP} = Macrocells in low-power mode

MC = Total number of macrocells used

f = Clock frequency (MHz)

Figure 1 shows a typical calculation for the XC9572 device.



DS065_01_110501

Figure 1: Typical I_{CC} vs. Frequency for XC9572

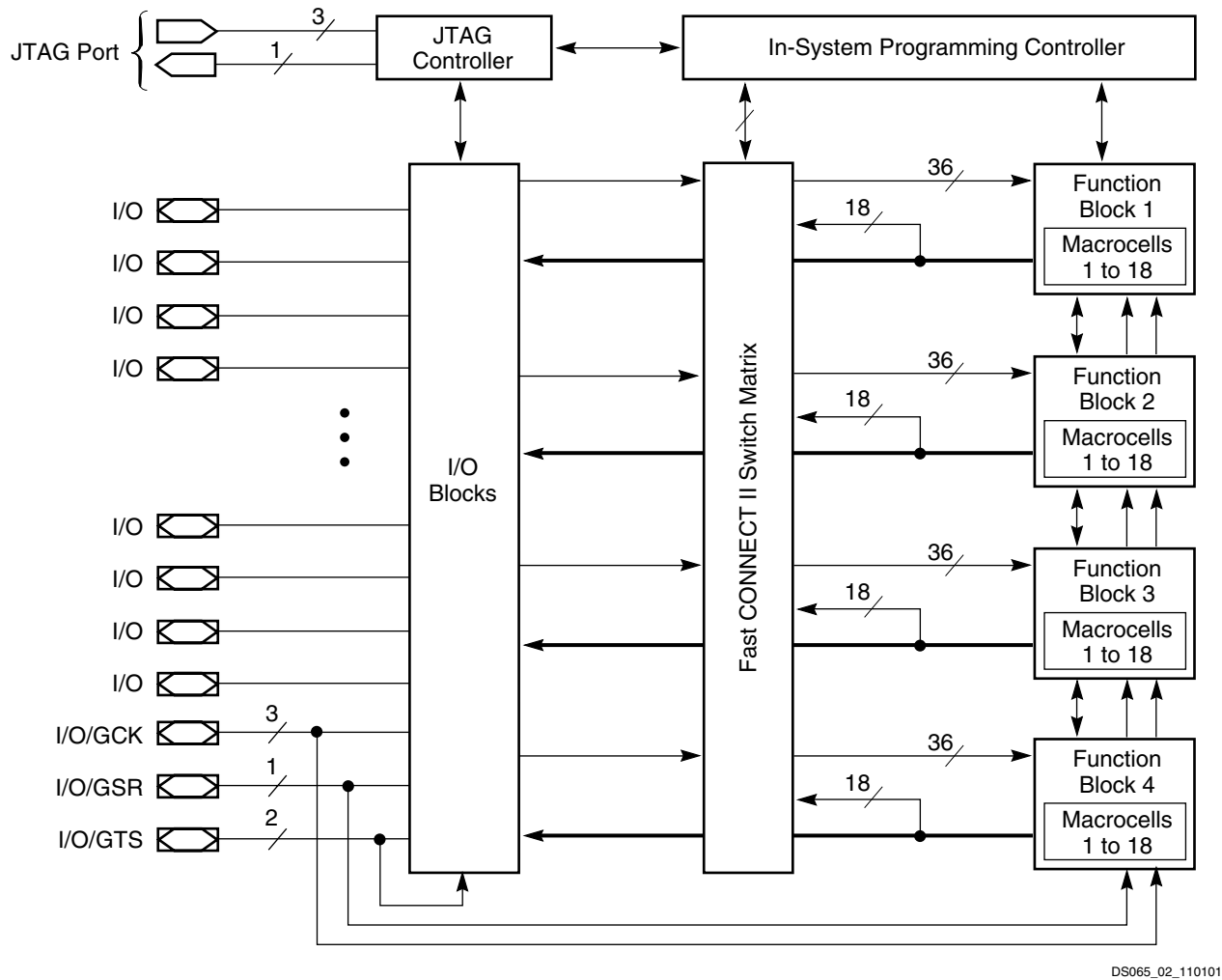


Figure 2: XC9572 Architecture

Function block outputs (indicated by the bold line) drive the I/O blocks directly.

Absolute Maximum Ratings

Symbol	Description	Value	Units
V_{CC}	Supply voltage relative to GND	-0.5 to 7.0	V
V_{IN}	Input voltage relative to GND	-0.5 to $V_{CC} + 0.5$	V
V_{TS}	Voltage applied to 3-state output	-0.5 to $V_{CC} + 0.5$	V
T_{STG}	Storage temperature (ambient)	-65 to +150	°C
T_J	Junction temperature	+150	°C

Notes:

- Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those listed under Operating Conditions is not implied. Exposure to Absolute Maximum Ratings conditions for extended periods of time may affect device reliability.

Recommended Operation Conditions

Symbol	Parameter		Min	Max	Units
V_{CCINT}	Supply voltage for internal logic and input buffers	Commercial $T_A = 0^{\circ}\text{C}$ to 70°C	4.75	5.25	V
		Industrial $T_A = -40^{\circ}\text{C}$ to $+85^{\circ}\text{C}$	4.5	5.5	
V_{CCIO}	Supply voltage for output drivers for 5V operation	Commercial $T_A = 0^{\circ}\text{C}$ to 70°C	4.75	5.25	V
		Industrial $T_A = -40^{\circ}\text{C}$ to $+85^{\circ}\text{C}$	4.5	5.5	
	Supply voltage for output drivers for 3.3V operation		3.0	3.6	
V_{IL}	Low-level input voltage		0	0.80	V
V_{IH}	High-level input voltage		2.0	$V_{CCINT} + 0.5$	V
V_O	Output voltage		0	V_{CCIO}	V

Quality and Reliability Characteristics

Symbol	Parameter	Min	Max	Units
T_{DR}	Data Retention	20	-	Years
N_{PE}	Program/Erase Cycles (Endurance)	10,000	-	Cycles

DC Characteristic Over Recommended Operating Conditions

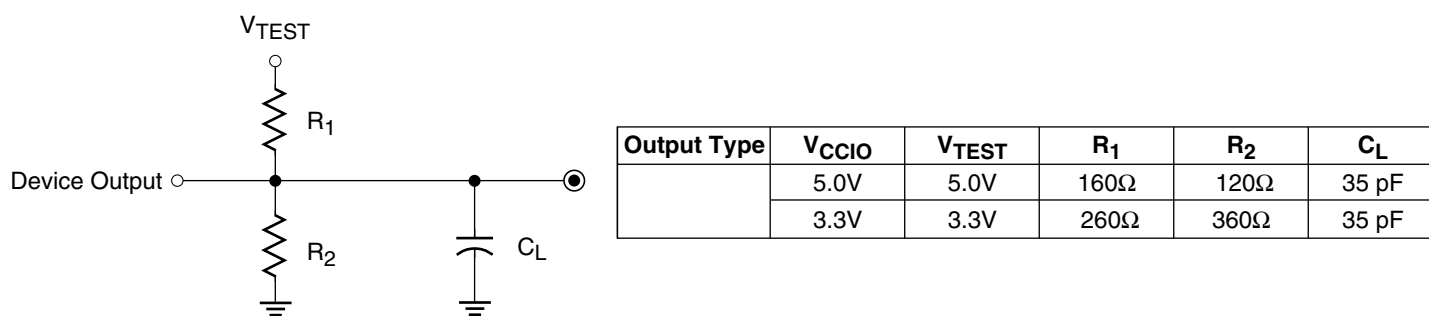
Symbol	Parameter	Test Conditions	Min	Max	Units
V_{OH}	Output high voltage for 5V outputs	$I_{OH} = -4.0\text{ mA}$, $V_{CC} = \text{Min}$	2.4	-	V
	Output high voltage for 3.3V outputs	$I_{OH} = -3.2\text{ mA}$, $V_{CC} = \text{Min}$	2.4	-	V
V_{OL}	Output low voltage for 5V outputs	$I_{OL} = 24\text{ mA}$, $V_{CC} = \text{Min}$	-	0.5	V
	Output low voltage for 3.3V outputs	$I_{OL} = 10\text{ mA}$, $V_{CC} = \text{Min}$	-	0.4	V
I_{IL}	Input leakage current	$V_{CC} = \text{Max}$ $V_{IN} = \text{GND or } V_{CC}$	-	± 10	μA
I_{IH}	I/O high-Z leakage current	$V_{CC} = \text{Max}$ $V_{IN} = \text{GND or } V_{CC}$	-	± 10	μA
C_{IN}	I/O capacitance	$V_{IN} = \text{GND}$ $f = 1.0\text{ MHz}$	-	10	pF
I_{CC}	Operating supply current (low power mode, active)	$V_I = \text{GND}$, No load $f = 1.0\text{ MHz}$	65 (Typical)		mA

AC Characteristics

Symbol	Parameter	XC9572-7		XC9572-10		XC9572-15		Units
		Min	Max	Min	Max	Min	Max	
T_{PD}	I/O to output valid	-	7.5	-	10.0	-	15.0	ns
T_{SU}	I/O setup time before GCK	4.5	-	6.0	-	8.0	-	ns
T_H	I/O hold time after GCK	0	-	0	-	0	-	ns
T_{CO}	GCK to output valid	-	4.5	-	6.0	-	8.0	ns
$f_{CNT}^{(1)}$	16-bit counter frequency	125.0	-	111.1	-	95.2	-	MHz
$f_{SYSTEM}^{(2)}$	Multiple FB internal operating frequency	83.3	-	66.7	-	55.6	-	MHz
T_{PSU}	I/O setup time before p-term clock input	0.5	-	2.0	-	4.0	-	ns
T_{PH}	I/O hold time after p-term clock input	4.0	-	4.0	-	4.0	-	ns
T_{PCO}	P-term clock output valid	-	8.5	-	10.0	-	12.0	ns
T_{OE}	GTS to output valid	-	5.5	-	6.0	-	11.0	ns
T_{OD}	GTS to output disable	-	5.5	-	6.0	-	11.0	ns
T_{POE}	Product term OE to output enabled	-	9.5	-	10.0	-	14.0	ns
T_{POD}	Product term OE to output disabled	-	9.5	-	10.0	-	14.0	ns
T_{WLH}	GCK pulse width (High or Low)	4.0	-	4.5	-	5.5	-	ns
T_{APRPW}	Asynchronous preset/reset pulse width (High or Low)	7.0	-	7.5	-	8.0	-	ns

Notes:

- f_{CNT} is the fastest 16-bit counter frequency available, using the local feedback when applicable.
 f_{CNT} is also the Export Control Maximum flip-flop toggle rate, f_{TOG} .
- f_{SYSTEM} is the internal operating frequency for general purpose system designs spanning multiple FBs.



DS067_03_110101

Figure 3: AC Load Circuit

Internal Timing Parameters

Symbol	Parameter	XC9572-7		XC9572-10		XC9572-15		Units
		Min	Max	Min	Max	Min	Max	
Buffer Delays								
T _{IN}	Input buffer delay	-	2.5	-	3.5	-	4.5	ns
T _{GCK}	GCK buffer delay	-	1.5	-	2.5	-	3.0	ns
T _{GSR}	GSR buffer delay	-	4.5	-	6.0	-	7.5	ns
T _{GTS}	GTS buffer delay	-	5.5	-	6.0	-	11.0	ns
T _{OUT}	Output buffer delay	-	2.5	-	3.0	-	4.5	ns
T _{EN}	Output buffer enable/disable delay	-	0	-	0	-	0	ns
Product Term Control Delays								
T _{PTCK}	Product term clock delay	-	3.0	-	3.0	-	2.5	ns
T _{PTSR}	Product term set/reset delay	-	2.0	-	2.5	-	3.0	ns
T _{PTTS}	Product term 3-state delay	-	4.5	-	3.5	-	5.0	ns
Internal Register and Combinatorial Delays								
T _{PDI}	Combinatorial logic propagation delay	-	0.5	-	1.0	-	3.0	ns
T _{SUI}	Register setup time	1.5	-	2.5	-	3.5	-	ns
T _{HI}	Register hold time	3.0	-	3.5	-	4.5	-	ns
T _{COI}	Register clock to output valid time	-	0.5	-	0.5	-	0.5	ns
T _{AOI}	Register async. S/R to output delay	-	6.5	-	7.0	-	8.0	ns
T _{RAI}	Register async. S/R recover before clock	7.5	-	10.0	-	10.0	-	ns
T _{LOGI}	Internal logic delay	-	2.0	-	2.5	-	3.0	ns
T _{LOGILP}	Internal low power logic delay	-	10.0	-	11.0	-	11.5	ns
Feedback Delays								
T _F	FastCONNECT feedback delay	-	8.0	-	9.5	-	11.0	ns
T _{LF}	Function block local feedback delay	-	4.0	-	3.5	-	3.5	ns
Time Adders								
T _{PTA} ⁽¹⁾	Incremental product term allocator delay	-	1.0	-	1.0	-	1.0	ns
T _{SLEW}	Slew-rate limited delay	-	4.0	-	4.5	-	5.0	ns

Notes:

- T_{PTA} is multiplied by the span of the function as defined in the XC9500 family data sheet.

XC9572 I/O Pins

Function Block	Macro-cell	PC44	PC84	PQ100	TQ100	BScan Order	Function Block	Macro-cell	PC44	PC84	PQ100	TQ100	BScan Order
1	1	—	4	18	16	213	3	1	—	25	43	41	105
1	2	1	1	15	13	210	3	2	11	17	34	32	102
1	3	—	6	20	18	207	3	3	—	31	51	49	99
1	4	—	7	22	20	204	3	4	—	32	52	50	96
1	5	2	2	16	14	201	3	5	12	19	37	35	93
1	6	3	3	17	15	198	3	6	—	34	55	53	90
1	7	—	11	27	25	195	3	7	—	35	56	54	87
1	8	4	5	19	17	192	3	8	13	21	39	37	84
1	9	5 ^[1]	9 ^[1]	24 ^[1]	22 ^[1]	189	3	9	14	26	44	42	81
1	10	—	13	30	28	186	3	10	—	40	62	60	78
1	11	6 ^[1]	10 ^[1]	25 ^[1]	23 ^[1]	183	3	11	18	33	54	52	75
1	12	—	18	35	33	180	3	12	—	41	63	61	72
1	13	—	20	38	36	177	3	13	—	43	65	63	69
1	14	7 ^[1]	12 ^[1]	29 ^[1]	27 ^[1]	174	3	14	19	36	57	55	66
1	15	8	14	31	29	171	3	15	20	37	58	56	63
1	16	—	23	41	39	168	3	16	—	45	67	65	60
1	17	9	15	32	30	165	3	17	22	39	60	58	57
1	18	—	24	42	40	162	3	18	—	—	61	59	54
2	1	—	63	89	87	159	4	1	—	46	68	66	51
2	2	35	69	96	94	156	4	2	24	44	66	64	48
2	3	—	67	93	91	153	4	3	—	51	73	71	45
2	4	—	68	95	93	150	4	4	—	52	74	72	42
2	5	36	70	97	95	147	4	5	25	47	69	67	39
2	6	37	71	98	96	144	4	6	—	54	78	76	36
2	7	—	76 ^[2]	5 ^[2]	3 ^[2]	141	4	7	—	55	79	77	33
2	8	38	72	99	97	138	4	8	26	48	70	68	30
2	9	39 ^[1]	74 ^[1]	1 ^[1]	99 ^[1]	135	4	9	27	50	72	70	27
2	10	—	75	3	1	132	4	10	—	57	83	81	24
2	11	40 ^[1]	77 ^[1]	6 ^[1]	4 ^[1]	129	4	11	28	53	76	74	21
2	12	—	79	8	6	126	4	12	—	58	84	82	18
2	13	—	80	10	8	123	4	13	—	61	87	85	15
2	14	42 ^[3]	81 ^[3]	11 ^[3]	9 ^[3]	120	4	14	29	56	80	78	12
2	15	43	83	13	11	117	4	15	33	65	91	89	9
2	16	—	82	12	10	114	4	16	—	62	88	86	6
2	17	44	84	14	12	111	4	17	34	66	92	90	3
2	18	—	—	94	92	108	4	18	—	—	81	79	0

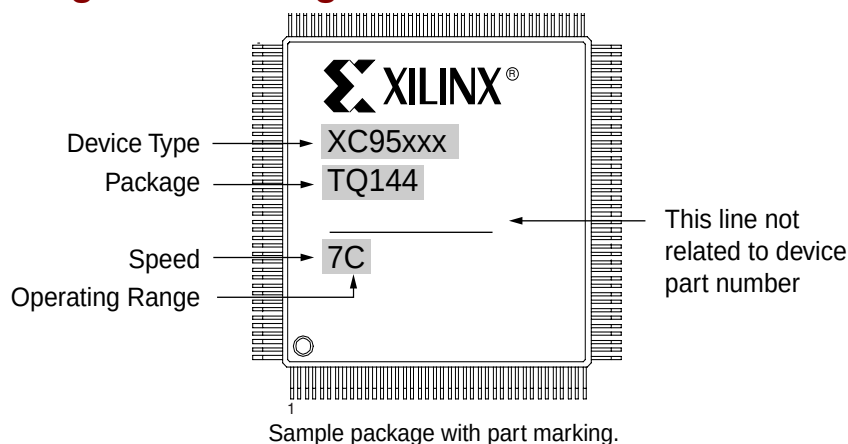
Notes:

1. Global control pinN.
2. Global control pin GTS1 for PC84, PQ100, and TQ100.
3. Global control pin GTS1 for PC44.

XC9572 Global, JTAG and Power Pins

Pin Type	PC44	PC84	PQ100	TQ100
I/O/GCK1	5	9	24	22
I/O/GCK2	6	10	25	23
I/O/GCK3	7	12	29	27
I/O/GTS1	42	76	5	3
I/O/GTS2	40	77	6	4
I/O/GSR	39	74	1	99
TCK	17	30	50	48
TDI	15	28	47	45
TDO	30	59	85	83
TMS	16	29	49	47
V _{CCINT} 5V	21,41	38,73,78	7,59,100	5,57,98
V _{CCIO} 3.3V/5V	32	22,64	28,40,53,90	26,38,51,88
GND	10,23,31	8,16,27,42, 49,60	2,23,33,46,64,71, 77,86	100,21,31,44,62,69, 75, 84
No Connects	-	-	4,9,21,26,36,45,48, 75, 82	2,7,19,24,34,43,46, 73, 80

Device Part Marking and Ordering Combination Information



Device Ordering and Part Marking Number	Speed (pin-to-pin delay)	Pkg. Symbol	No. of Pins	Package Type	Operating Range ⁽¹⁾
XC9572-7PC44C	7.5 ns	PC44	44-pin	Plastic Lead Chip Carrier (PLCC)	C
XC9572-7PCG44C	7.5 ns	PCG44	44-pin	Plastic Lead Chip Carrier (PLCC); Pb-Free	C
XC9572-7PC84C	7.5 ns	PC84	84-pin	Plastic Lead Chip Carrier (PLCC)	C
XC9572-7PCG84C	7.5 ns	PCG84	84-pin	Plastic Lead Chip Carrier (PLCC); Pb-Free	C
XC9572-7PQ100C	7.5 ns	PQ100	100-pin	Plastic Quad Flat Pack (PQFP)	C
XC9572-7PQG100C	7.5 ns	PQG100	100-pin	Plastic Quad Flat Pack (PQFP); Pb-Free	C
XC9572-7TQ100C	7.5 ns	TQ100	100-pin	Thin Quad Flat Pack (TQFP)	C
XC9572-7TQG100C	7.5 ns	TQG100	100-pin	Thin Quad Flat Pack (TQFP); Pb-Free	C
XC9572-10PC44C	10 ns	PC44	44-pin	Plastic Lead Chip Carrier (PLCC)	C
XC9572-10PCG44C	10 ns	PCG44	44-pin	Plastic Lead Chip Carrier (PLCC); Pb-Free	C
XC9572-10PC84C	10 ns	PC84	84-pin	Plastic Lead Chip Carrier (PLCC)	C
XC9572-10PCG84C	10 ns	PCG84	84-pin	Plastic Lead Chip Carrier (PLCC); Pb-Free	C
XC9572-10PQ100C	10 ns	PQ100	100-pin	Plastic Quad Flat Pack (PQFP)	C
XC9572-10PQG100C	10 ns	PQG100	100-pin	Plastic Quad Flat Pack (PQFP); Pb-Free	C
XC9572-10TQ100C	10 ns	TQ100	100-pin	Thin Quad Flat Pack (TQFP)	C
XC9572-10TQG100C	10 ns	TQG100	100-pin	Thin Quad Flat Pack (TQFP); Pb-Free	C
XC9572-10PC44I	10 ns	PC44	44-pin	Plastic Lead Chip Carrier (PLCC)	I
XC9572-10PCG44I	10 ns	PCG44	44-pin	Plastic Lead Chip Carrier (PLCC); Pb-Free	I
XC9572-10PC84I	10 ns	PC84	84-pin	Plastic Lead Chip Carrier (PLCC)	I
XC9572-10PCG84I	10 ns	PCG84	84-pin	Plastic Lead Chip Carrier (PLCC); Pb-Free	I
XC9572-10PQ100I	10 ns	PQ100	100-pin	Plastic Quad Flat Pack (PQFP)	I
XC9572-10PQG100I	10 ns	PQG100	100-pin	Plastic Quad Flat Pack (PQFP); Pb-Free	I
XC9572-10TQ100I	10 ns	TQ100	100-pin	Thin Quad Flat Pack (TQFP)	I
XC9572-10TQG100I	10 ns	TQG100	100-pin	Thin Quad Flat Pack (TQFP); Pb-Free	I
XC9572-15PC44C	15 ns	PC44	44-pin	Plastic Lead Chip Carrier (PLCC)	C
XC9572-15PCG44C	15 ns	PCG44	44-pin	Plastic Lead Chip Carrier (PLCC); Pb-Free	C
XC9572-15PC84C	15 ns	PC84	84-pin	Plastic Lead Chip Carrier (PLCC)	C
XC9572-15PCG84C	15 ns	PCG84	84-pin	Plastic Lead Chip Carrier (PLCC); Pb-Free	C
XC9572-15PQ100C	15 ns	PQ100	100-pin	Plastic Quad Flat Pack (PQFP)	C
XC9572-15PQG100C	15 ns	PQG100	100-pin	Plastic Quad Flat Pack (PQFP); Pb-Free	C
XC9572-15TQ100C	15 ns	TQ100	100-pin	Thin Quad Flat Pack (TQFP)	C
XC9572-15TQG100C	15 ns	TQG100	100-pin	Thin Quad Flat Pack (TQFP); Pb-Free	C
XC9572-15PC44I	15 ns	PC44	44-pin	Plastic Lead Chip Carrier (PLCC)	I

Device Ordering and Part Marking Number	Speed (pin-to-pin delay)	Pkg. Symbol	No. of Pins	Package Type	Operating Range ⁽¹⁾
XC9572-15PCG44I	15 ns	PCG44	44-pin	Plastic Lead Chip Carrier (PLCC); Pb-Free	I
XC9572-15PC84I	15 ns	PC84	84-pin	Plastic Lead Chip Carrier (PLCC)	I
XC9572-15PCG84I	15 ns	PCG84	84-pin	Plastic Lead Chip Carrier (PLCC); Pb-Free	I
XC9572-15PQG100I	15 ns	PQG100	100-pin	Plastic Quad Flat Pack (PQFP)	I
XC9572-15PQG100I	15 ns	PQG100	100-pin	Plastic Quad Flat Pack (PQFP); Pb-Free	I
XC9572-15TQ100I	15 ns	TQ100	100-pin	Thin Quad Flat Pack (TQFP)	I
XC9572-15TQG100I	15 ns	TQG100	100-pin	Thin Quad Flat Pack (TQFP); Pb-Free	I

Notes:

1. C = Commercial: $T_A = 0^\circ$ to $+70^\circ\text{C}$; I = Industrial: $T_A = -40^\circ$ to $+85^\circ\text{C}$

Warranty Disclaimer

THESE PRODUCTS ARE SUBJECT TO THE TERMS OF THE XILINX LIMITED WARRANTY WHICH CAN BE VIEWED AT <http://www.xilinx.com/warranty.htm>. THIS LIMITED WARRANTY DOES NOT EXTEND TO ANY USE OF THE PRODUCTS IN AN APPLICATION OR ENVIRONMENT THAT IS NOT WITHIN THE SPECIFICATIONS STATED ON THE THEN-CURRENT XILINX DATA SHEET FOR THE PRODUCTS. PRODUCTS ARE NOT DESIGNED TO BE FAIL-SAFE AND ARE NOT WARRANTED FOR USE IN APPLICATIONS THAT POSE A RISK OF PHYSICAL HARM OR LOSS OF LIFE. USE OF PRODUCTS IN SUCH APPLICATIONS IS FULLY AT THE RISK OF CUSTOMER SUBJECT TO APPLICABLE LAWS AND REGULATIONS.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/04/98	3.0	Update AC characteristics and internal parameters.
06/18/03	4.0	Updated format.
08/21/03	4.1	Updated Package Device Marking Pin 1 orientation.
04/15/05	4.2	Added asynchronous preset/reset pulse width specification (T_{APRPW})
04/03/06	4.3	Added Warranty Disclaimer. Added Pb-Free package information.

Features

- High-performance
 - 5 ns pin-to-pin logic delays on all pins
 - f_{CNT} to 125 MHz
- Large density range
 - 36 to 288 macrocells with 800 to 6,400 usable gates
- 5V in-system programmable
 - Endurance of 10,000 program/erase cycles
 - Program/erase over full commercial voltage and temperature range
- Enhanced pin-locking architecture
- Flexible 36V18 Function Block
 - 90 product terms drive any or all of 18 macrocells within Function Block
 - Global and product term clocks, output enables, set and reset signals
 - Extensive IEEE Std 1149.1 boundary-scan (JTAG) support
 - Programmable power reduction mode in each macrocell
 - Slew rate control on individual outputs
 - User programmable ground pin capability
 - Extended pattern security features for design protection
 - High-drive 24 mA outputs
 - 3.3V or 5V I/O capability

- Advanced CMOS 5V FastFLASH™ technology
- Supports parallel programming of multiple XC9500 devices

Family Overview

The XC9500 CPLD family provides advanced in-system programming and test capabilities for high performance, general purpose logic integration. All devices are in-system programmable for a minimum of 10,000 program/erase cycles. Extensive IEEE 1149.1 (JTAG) boundary-scan support is also included on all family members.

As shown in [Table 1](#), logic density of the XC9500 devices ranges from 800 to over 6,400 usable gates with 36 to 288 registers, respectively. Multiple package options and associated I/O capacity are shown in [Table 2](#). The XC9500 family is fully pin-compatible allowing easy design migration across multiple density options in a given package footprint.

The XC9500 architectural features address the requirements of in-system programmability. Enhanced pin-locking capability avoids costly board rework. An expanded JTAG instruction set allows version control of programming patterns and in-system debugging. In-system programming throughout the full device operating range and a minimum of 10,000 program/erase cycles provide worry-free reconfigurations and system field upgrades.

Advanced system features include output slew rate control and user-programmable ground pins to help reduce system noise. I/Os may be configured for 3.3V or 5V operation. All outputs provide 24 mA drive.

Table 1: XC9500 Device Family

	XC9536	XC9572	XC95108	XC95144	XC95216	XC95288
Macrocells	36	72	108	144	216	288
Usable Gates	800	1,600	2,400	3,200	4,800	6,400
Registers	36	72	108	144	216	288
T_{PD} (ns)	5	7.5	7.5	7.5	10	15
T_{SU} (ns)	3.5	4.5	4.5	4.5	6.0	8.0
T_{CO} (ns)	4.0	4.5	4.5	4.5	6.0	8.0
f_{CNT} (MHz) ⁽¹⁾	100	125	125	125	111.1	92.2
f_{SYSTEM} (MHz) ⁽²⁾	100	83.3	83.3	83.3	66.7	56.6

1. f_{CNT} = Operating frequency for 16-bit counters.

2. f_{SYSTEM} = Internal operating frequency for general purpose system designs spanning multiple FBs.

Table 2: Available Packages and Device I/O Pins (not including dedicated JTAG pins)

	XC9536	XC9572	XC95108	XC95144	XC95216	XC95288
44-Pin VQFP	34	-	-	-	-	-
44-Pin PLCC	34	34	-	-	-	-
48-Pin CSP	34	-	-	-	-	-
84-Pin PLCC	-	69	69	-	-	-
100-Pin TQFP	-	72	81	81	-	-
100-Pin PQFP	-	72	81	81	-	-
160-Pin PQFP	-	-	108	133	133	-
208-Pin HQFP	-	-	-	-	166	168
352-Pin BGA	-	-	-	-	166 ⁽²⁾	192

1. Most packages available in Pb-Free option. See individual data sheets for more details.

2. 352-pin BGA package is being discontinued for the XC95216. See [XC95216](#) for details.

Architecture Description

Each XC9500 device is a subsystem consisting of multiple Function Blocks (FBs) and I/O Blocks (IOBs) fully interconnected by the Fast CONNECT™ switch matrix. The IOB provides buffering for device inputs and outputs. Each FB

provides programmable logic capability with 36 inputs and 18 outputs. The Fast CONNECT switch matrix connects all FB outputs and input signals to the FB inputs. For each FB, 12 to 18 outputs (depending on package pin-count) and

associated output enable signals drive directly to the IOBs.
See Figure 1.

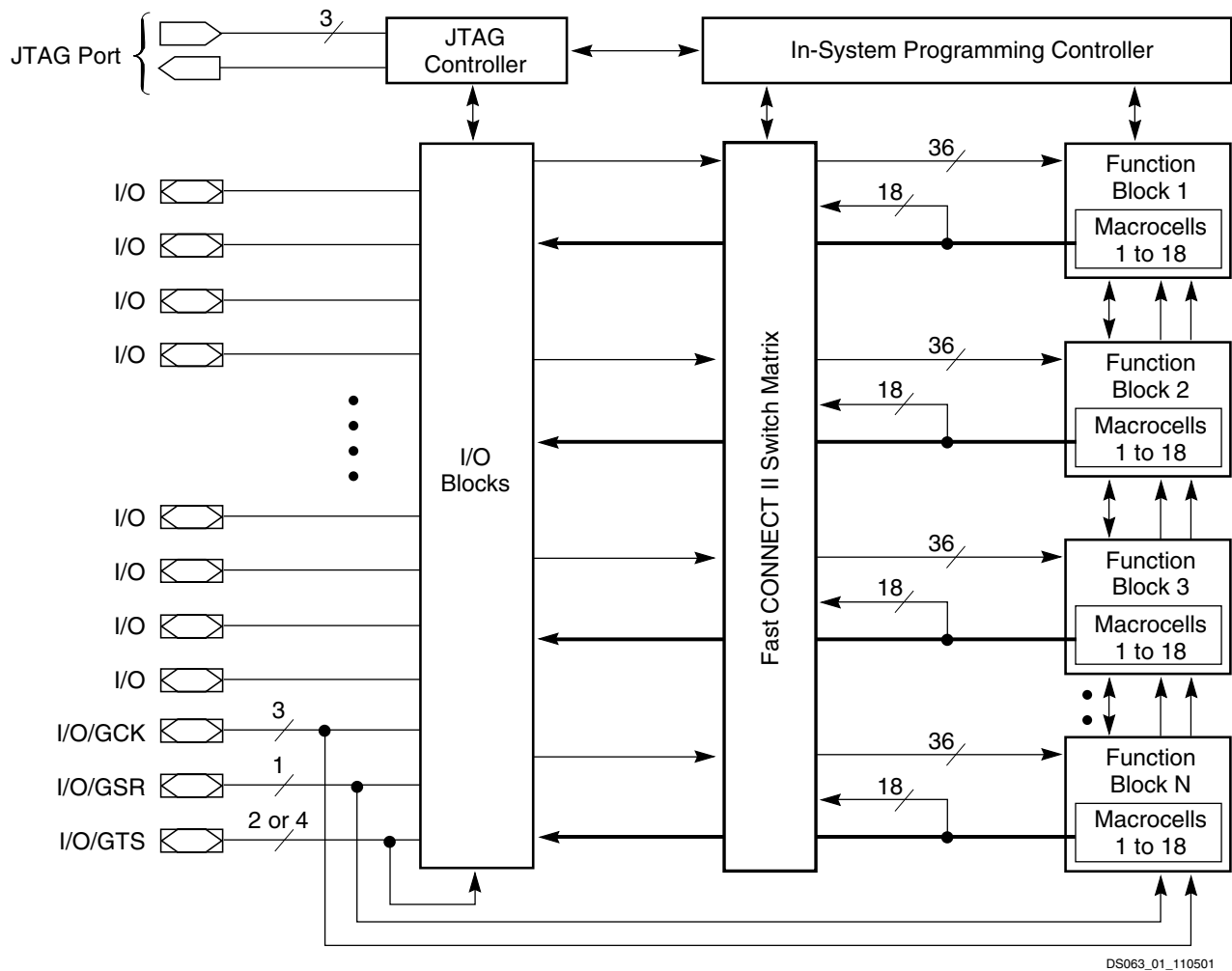


Figure 1: XC9500 Architecture

Note: Function block outputs (indicated by the bold lines) drive the I/O blocks directly.

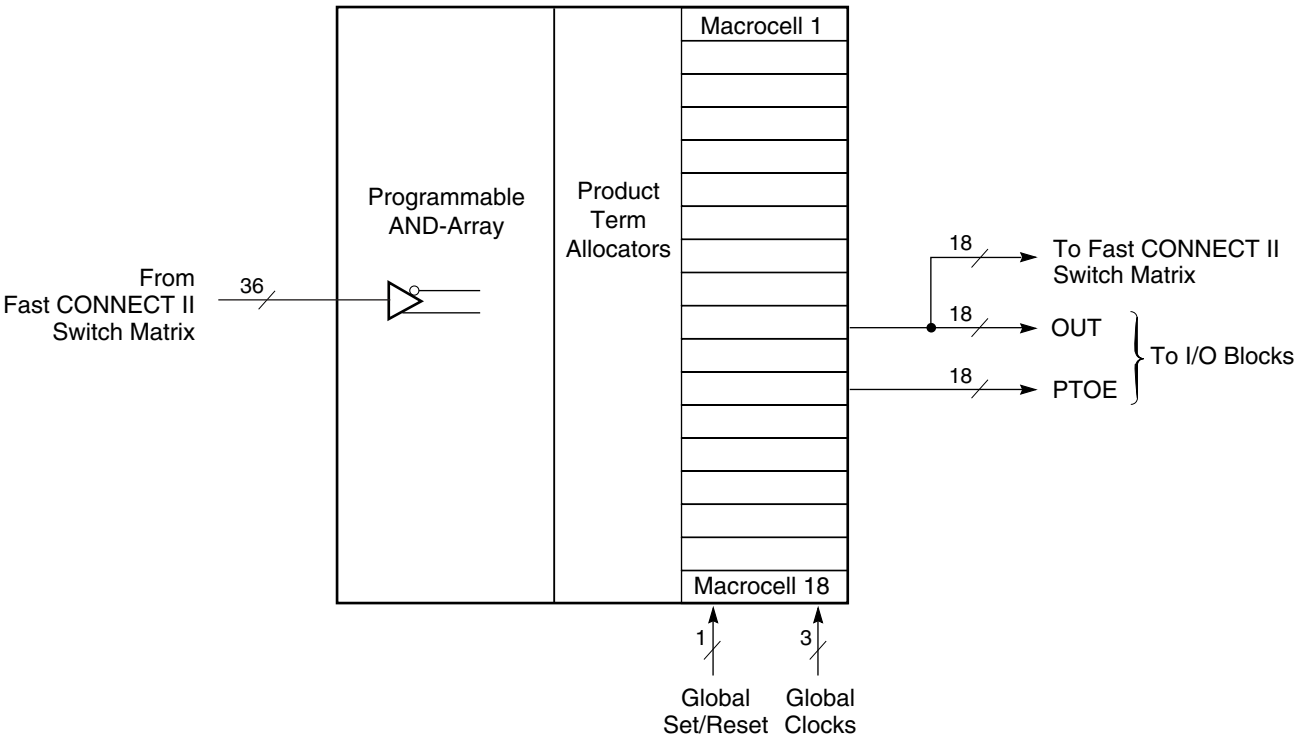
Function Block

Each Function Block, as shown in Figure 2, is comprised of 18 independent macrocells, each capable of implementing a combinatorial or registered function. The FB also receives global clock, output enable, and set/reset signals. The FB generates 18 outputs that drive the Fast CONNECT switch matrix. These 18 outputs and their corresponding output enable signals also drive the IOB.

Logic within the FB is implemented using a sum-of-products representation. Thirty-six inputs provide 72 true and complement signals into the programmable AND-array to form

90 product terms. Any number of these product terms, up to the 90 available, can be allocated to each macrocell by the product term allocator.

Each FB (except for the XC9536) supports local feedback paths that allow any number of FB outputs to drive into its own programmable AND-array without going outside the FB. These paths are used for creating very fast counters and state machines where all state registers are within the same FB.



DS063_02_110501

Figure 2: XC9500 Function Block

Macrocell

Each XC9500 macrocell may be individually configured for a combinatorial or registered function. The macrocell and associated FB logic is shown in **Figure 3**.

Five direct product terms from the AND-array are available for use as primary data inputs (to the OR and XOR gates) to implement combinatorial functions, or as control inputs including clock, set/reset, and output enable. The product

term allocator associated with each macrocell selects how the five direct terms are used.

The macrocell register can be configured as a D-type or T-type flip-flop, or it may be bypassed for combinatorial operation. Each register supports both asynchronous set and reset operations. During power-up, all user registers are initialized to the user-defined preload state (default to 0 if unspecified).

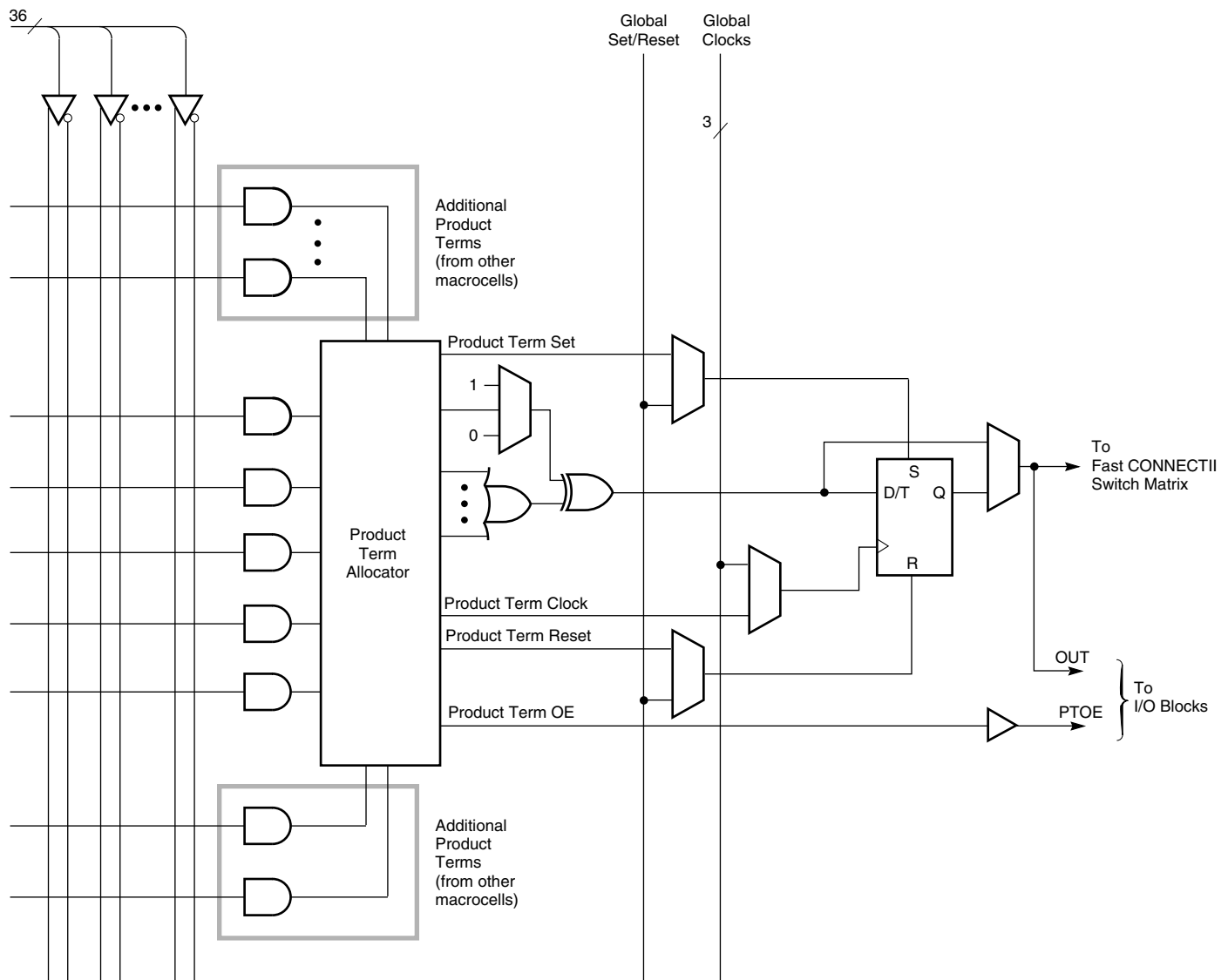
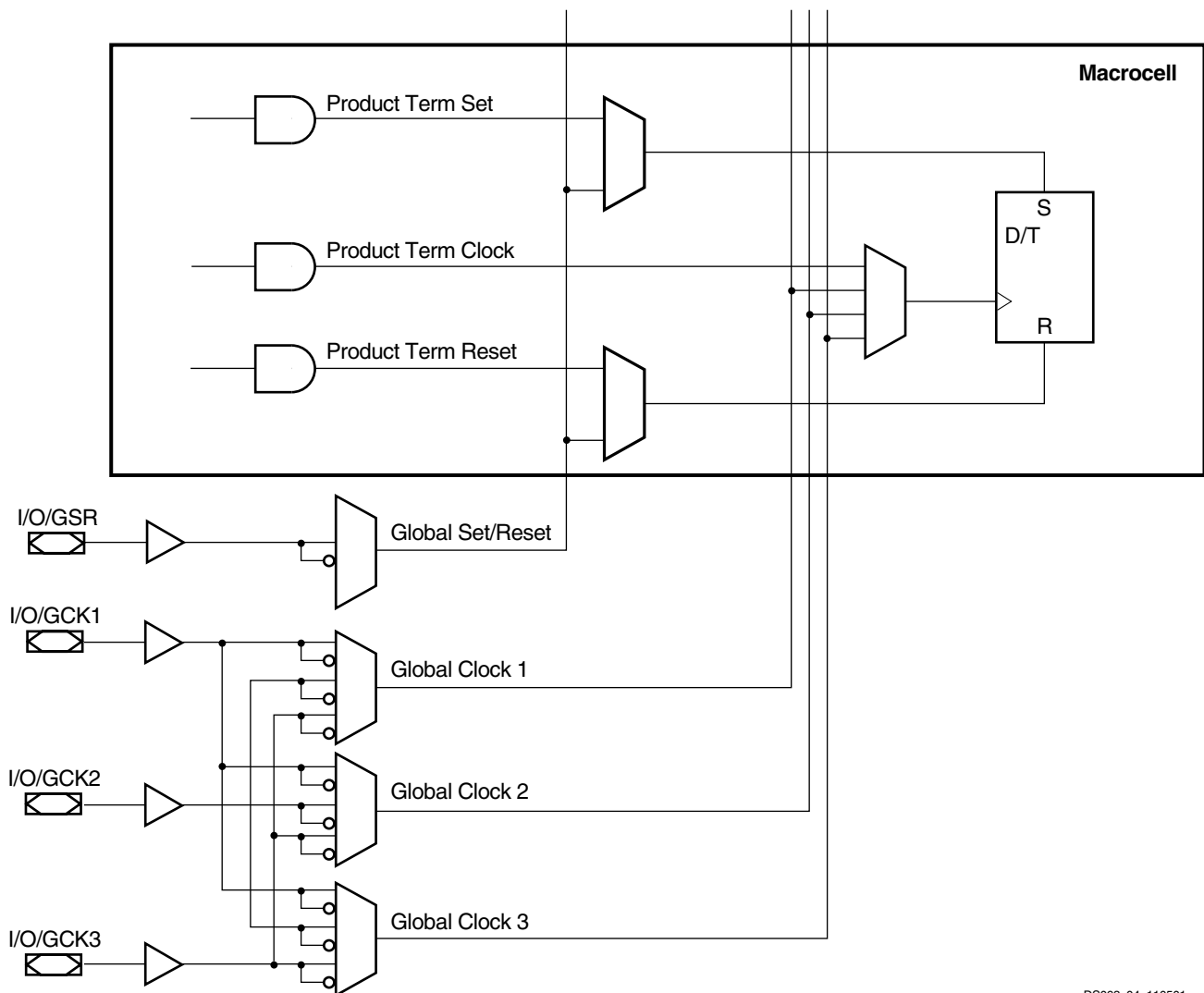


Figure 3: XC9500 Macrocell Within Function Block

All global control signals are available to each individual macrocell, including clock, set/reset, and output enable signals. As shown in **Figure 4**, the macrocell register clock originates from either of three global clocks or a product

term clock. Both true and complement polarities of a GCK pin can be used within the device. A GSR input is also provided to allow user registers to be set to a user-defined state.

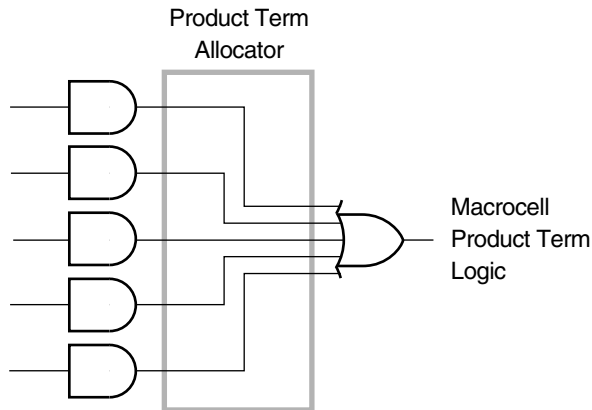


DS063_04_110501

Figure 4: Macrocell Clock and Set/Reset Capability

Product Term Allocator

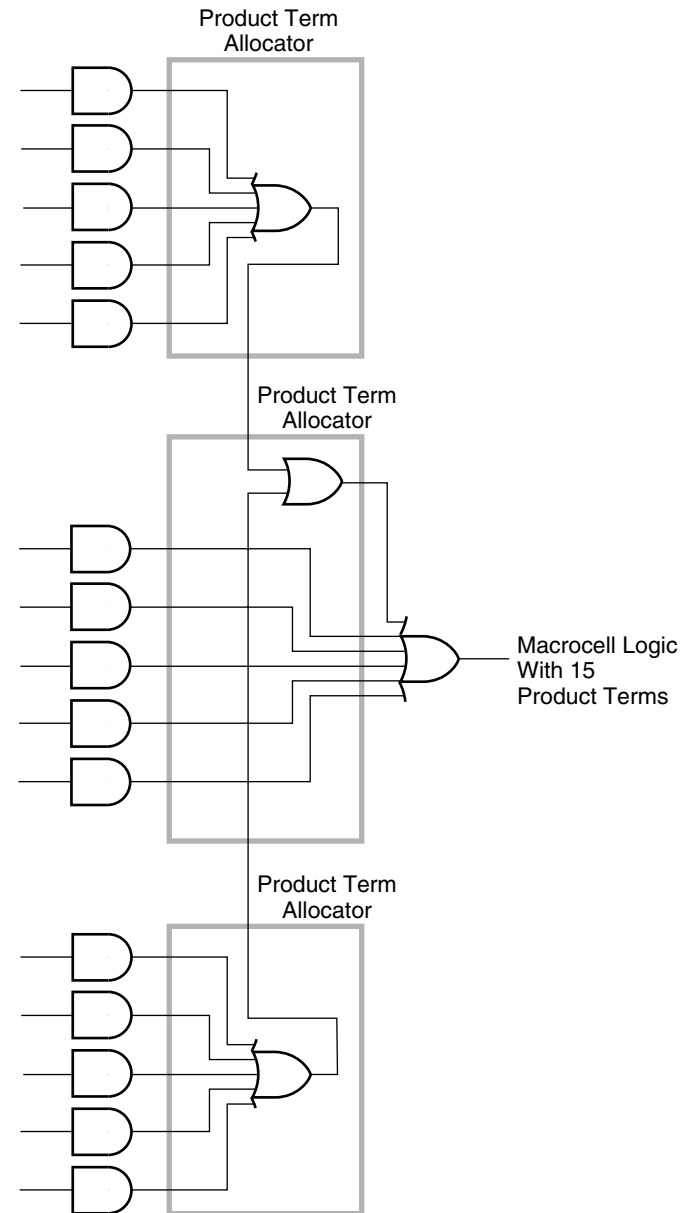
The product term allocator controls how the five direct product terms are assigned to each macrocell. For example, all five direct terms can drive the OR function as shown in Figure 5.



DS063_05_110501

Figure 5: Macrocell Logic Using Direct Product Term

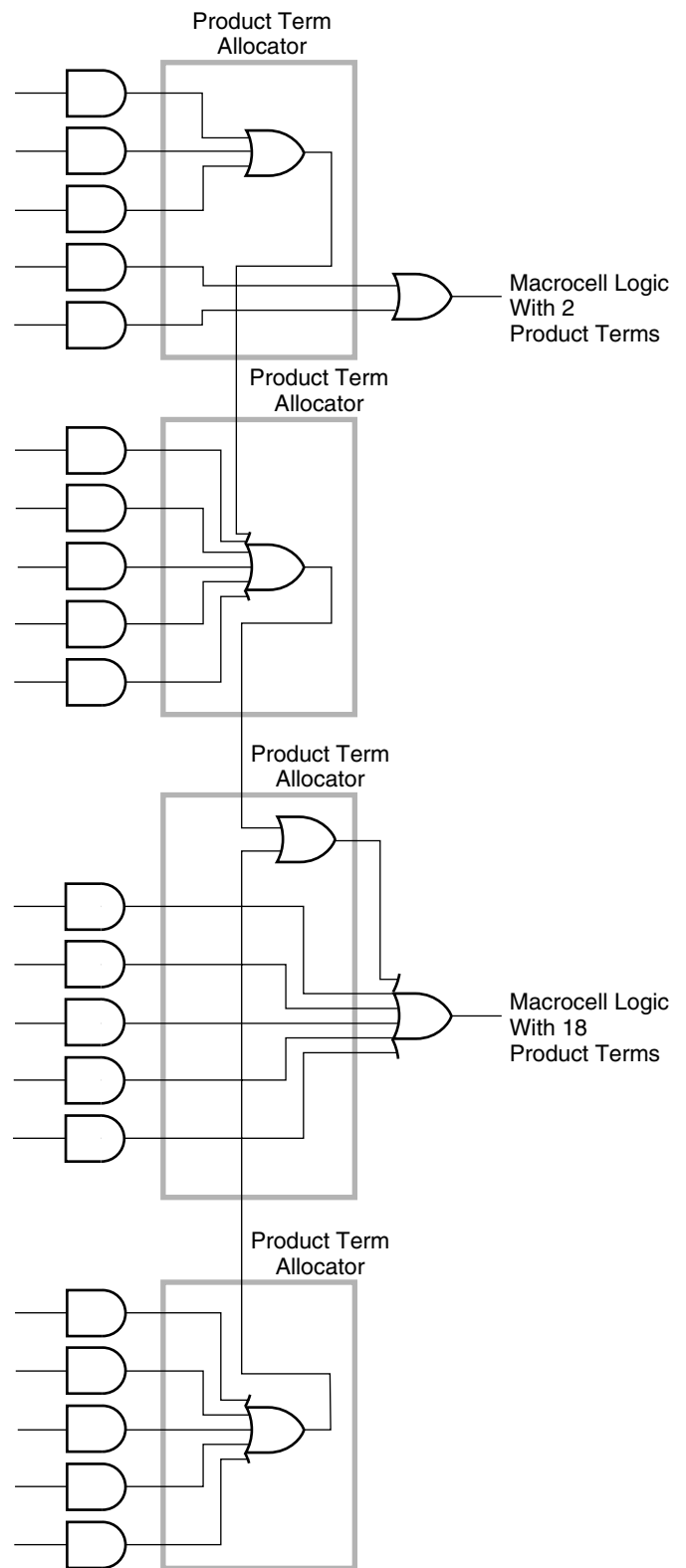
The product term allocator can re-assign other product terms within the FB to increase the logic capacity of a macrocell beyond five direct terms. Any macrocell requiring additional product terms can access uncommitted product terms in other macrocells within the FB. Up to 15 product terms can be available to a single macrocell with only a small incremental delay of T_{PTA} , as shown in Figure 6.



DS063_06_110501

Figure 6: Product Term Allocation With 15 Product Terms

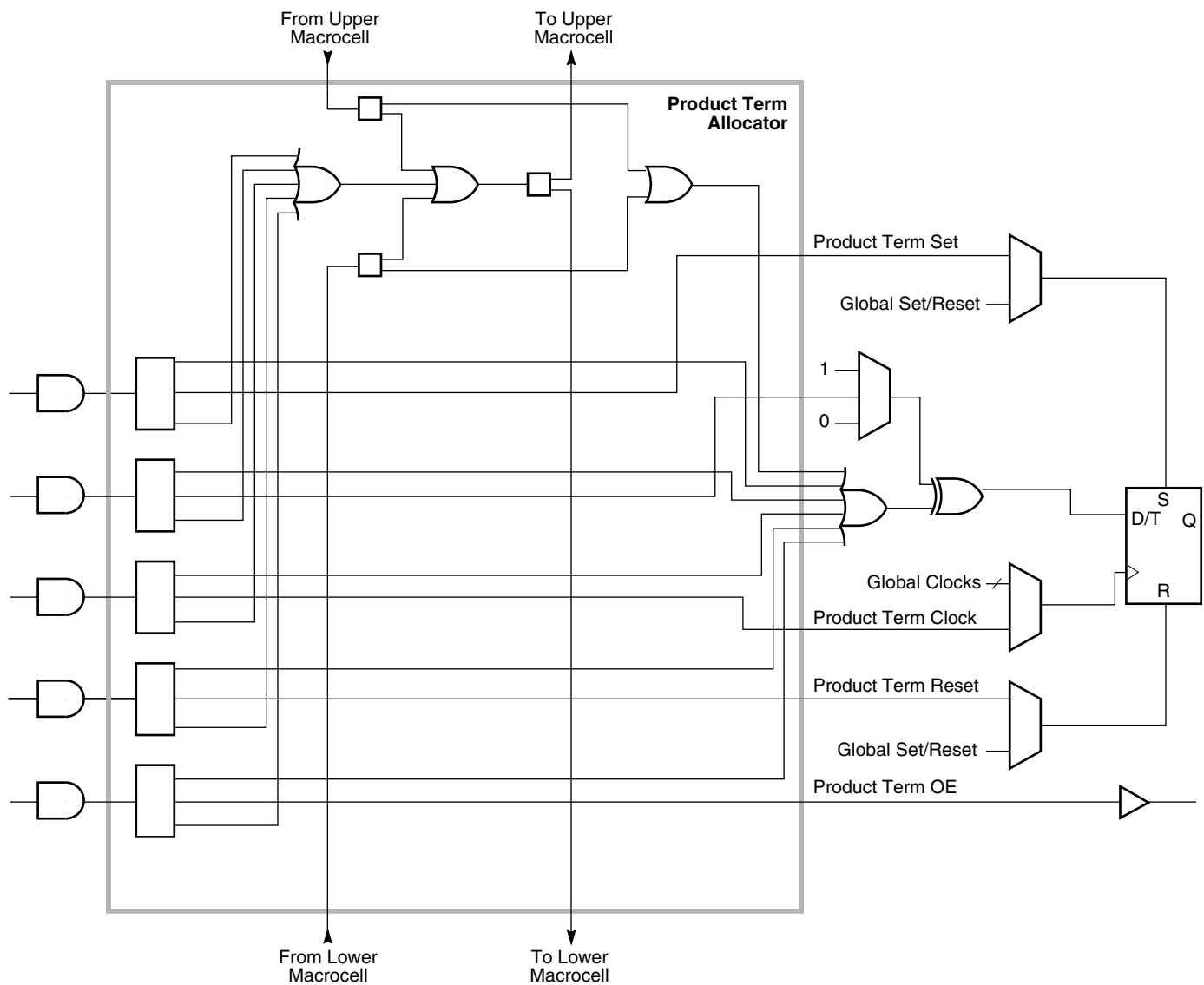
The product term allocator can re-assign product terms from any macrocell within the FB by combining partial sums of products over several macrocells, as shown in **Figure 7**. In this example, the incremental delay is only $2 \cdot T_{PTA}$. All 90 product terms are available to any macrocell, with a maximum incremental delay of $8 \cdot T_{PTA}$.



DS063_07_110501

Figure 7: Product Term Allocation Over Several Macrocells

The internal logic of the product term allocator is shown in **Figure 8**.



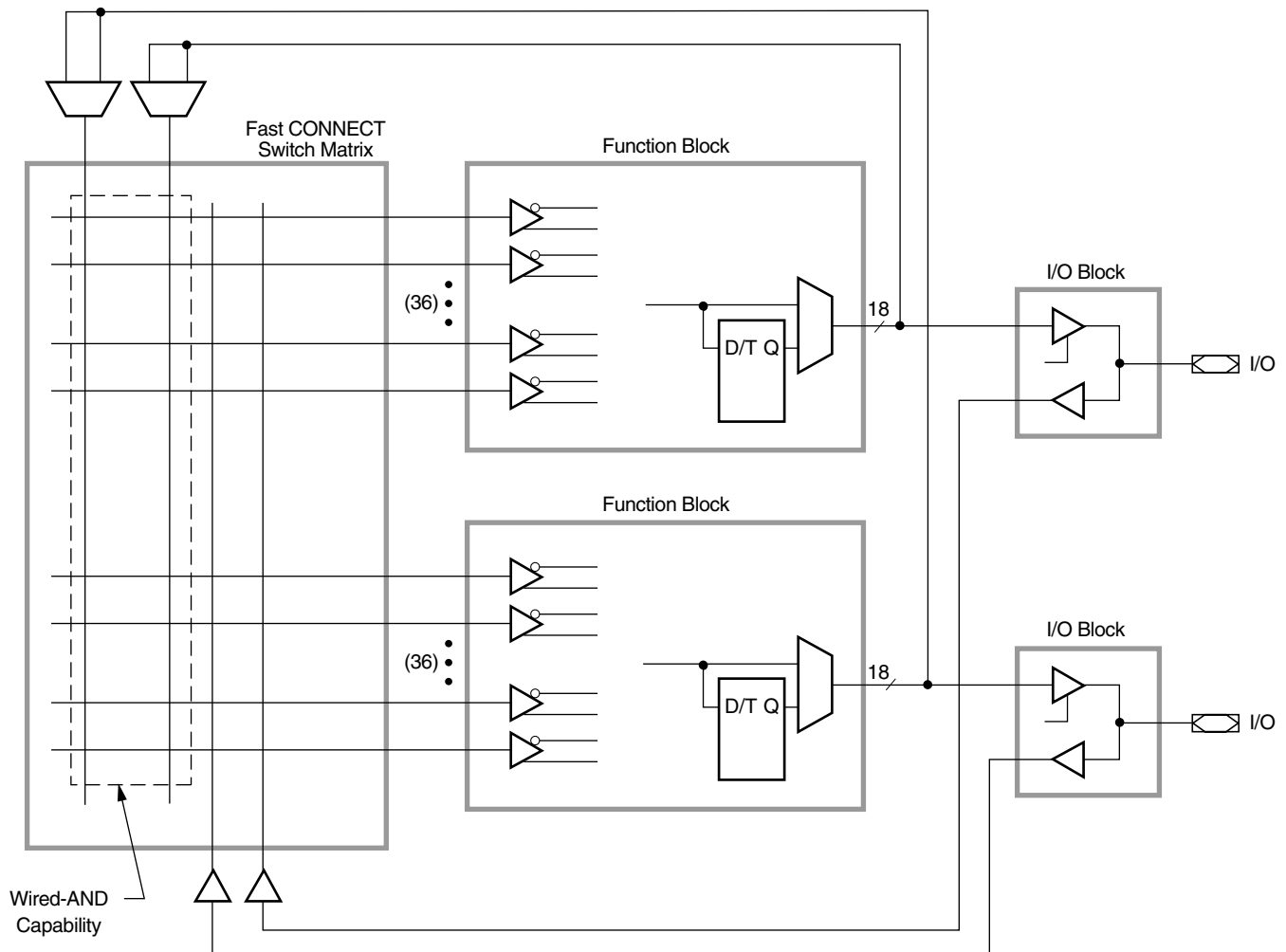
DS063_08_110501

Figure 8: Product Term Allocator Logic

Fast CONNECT Switch Matrix

The Fast CONNECT switch matrix connects signals to the FB inputs, as shown in **Figure 9**. All IOB outputs (corresponding to user pin inputs) and all FB outputs drive the Fast CONNECT matrix. Any of these (up to a FB fan-in limit of 36) may be selected, through user programming, to drive each FB with a uniform delay.

The Fast CONNECT switch matrix is capable of combining multiple internal connections into a single wired-AND output before driving the destination FB. This provides additional logic capability and increases the effective logic fan-in of the destination FB without any additional timing delay. This capability is available for internal connections originating from FB outputs only. It is automatically invoked by the development software where applicable.



DS063_09_110501

Figure 9: Fast CONNECT Switch Matrix

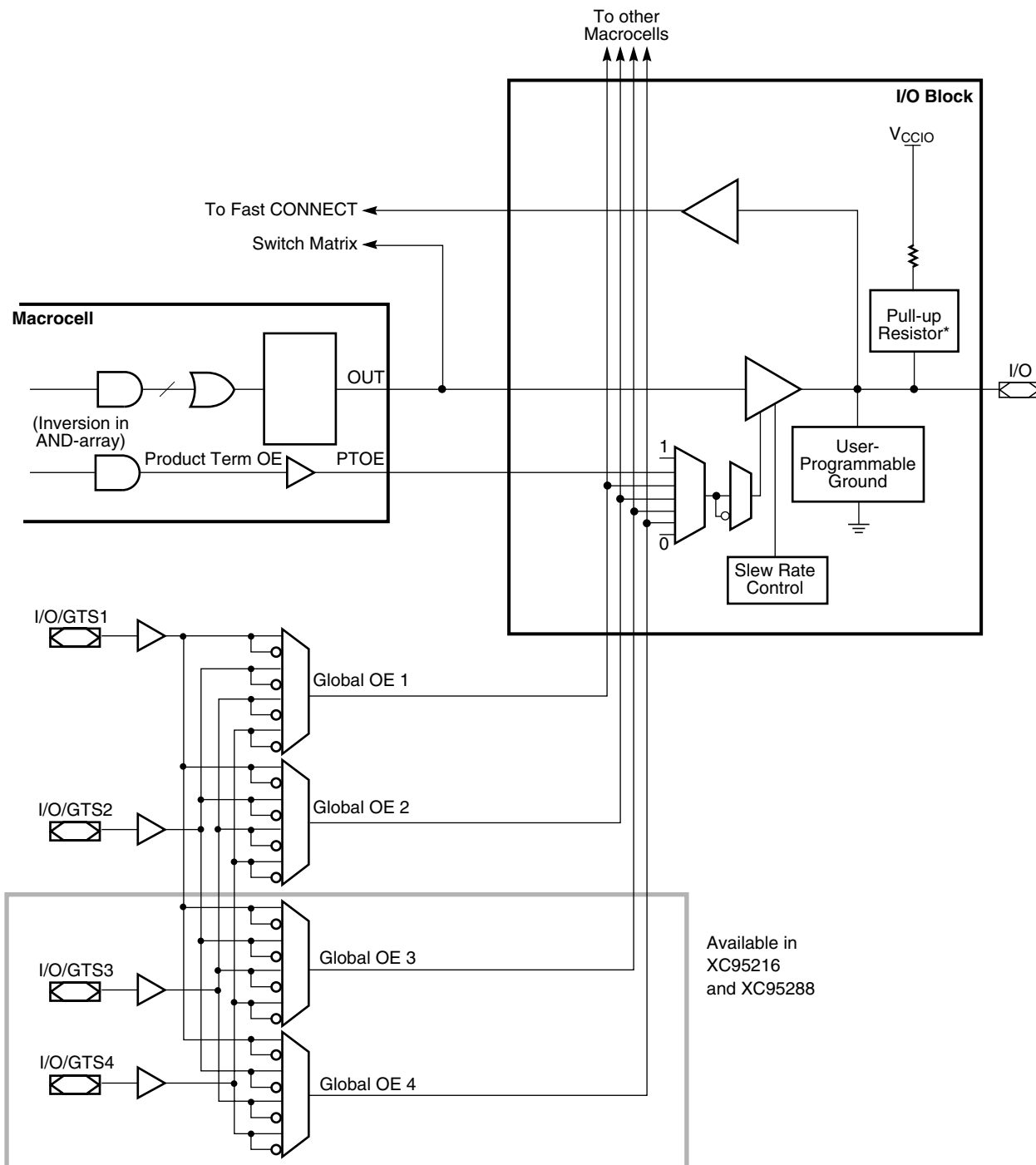
I/O Block

The I/O Block (IOB) interfaces between the internal logic and the device user I/O pins. Each IOB includes an input buffer, output driver, output enable selection multiplexer, and user programmable ground control. See Figure 10 for details.

The input buffer is compatible with standard 5V CMOS, 5V TTL, and 3.3V signal levels. The input buffer uses the internal

5V voltage supply (V_{CCINT}) to ensure that the input thresholds are constant and do not vary with the V_{CCIO} voltage.

The output enable may be generated from one of four options: a product term signal from the macrocell, any of the global OE signals, always [1], or always [0]. There are two global output enables for devices with up to 144 macrocells, and four global output enables for the rest of the devices. Both polarities of any of the global 3-state control (GTS) pins may be used within the device..



DS063_10_092203

Figure 10: I/O Block and Output Enable Capability

Each output has independent slew rate control. Output edge rates may be slowed down to reduce system noise (with an additional time delay of T_{SLEW}) through programming. See **Figure 11**.

Each IOB provides user programmable ground pin capability. This allows device I/O pins to be configured as additional ground pins. By tying strategically located programmable ground pins to the external ground connection, system noise generated from large numbers of simultaneous switching outputs may be reduced.

A control pull-up resistor (typically 10K ohms) is attached to each device I/O pin to prevent them from floating when the device is not in normal user operation. This resistor is active during device programming mode and system power-up. It is also activated for an erased device. The resistor is deactivated during normal operation.

The output driver is capable of supplying 24 mA output drive. All output drivers in the device may be configured for either 5V TTL levels or 3.3V levels by connecting the device output voltage supply (V_{CCIO}) to a 5V or 3.3V voltage supply.

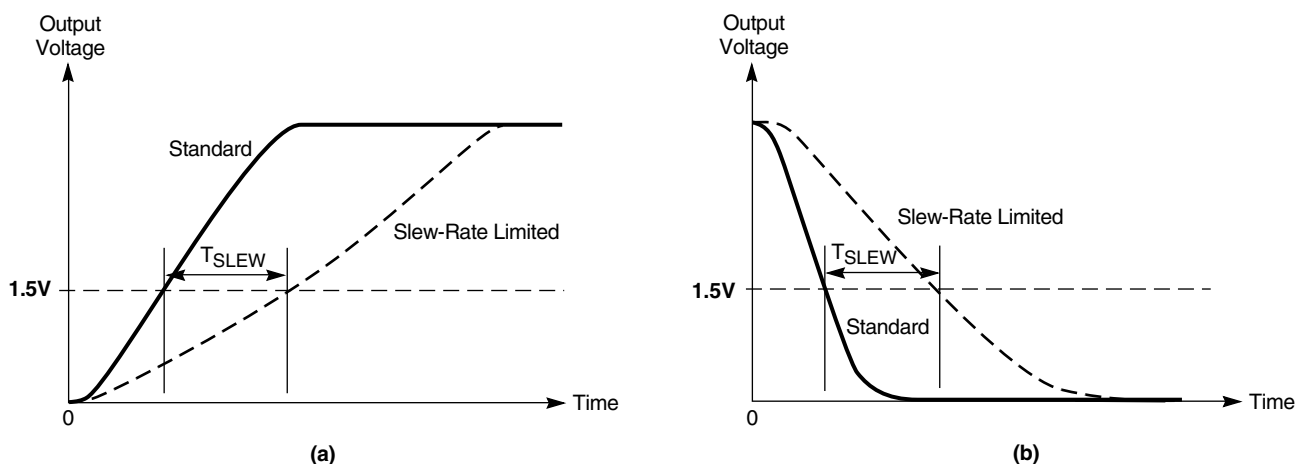
Figure 12 shows how the XC9500 device can be used in 5V only and mixed 3.3V/5V systems.

Pin-Locking Capability

The capability to lock the user defined pin assignments during design changes depends on the ability of the architecture to adapt to unexpected changes. The XC9500 devices have architectural features that enhance the ability to accept design changes while maintaining the same pinout.

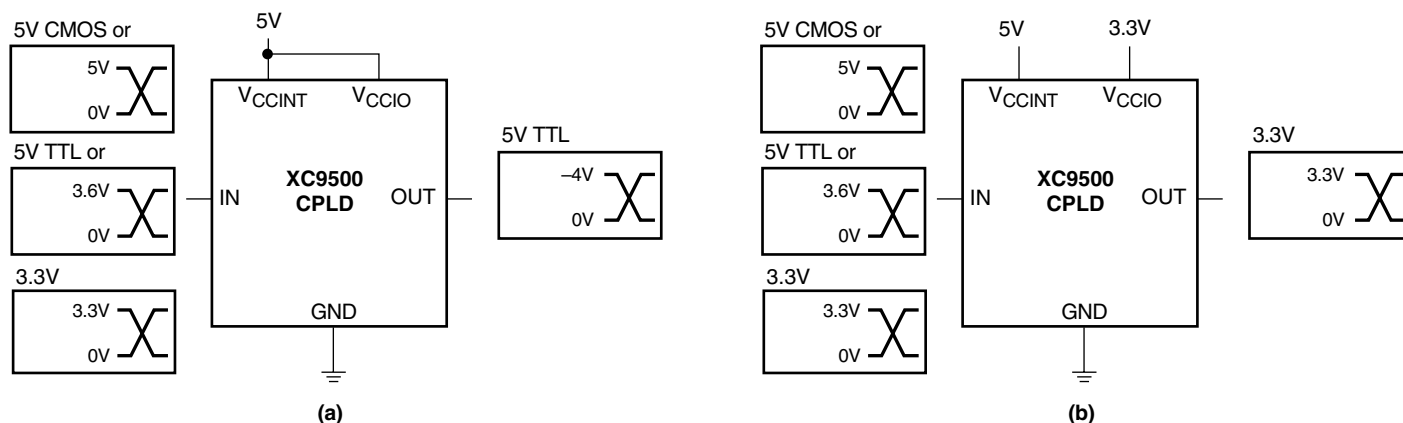
The XC9500 architecture provides maximum routing within the Fast CONNECT switch matrix, and incorporates a flexible Function Block that allows block-wide allocation of available product terms. This provides a high level of confidence of maintaining both input and output pin assignments for unexpected design changes.

For extensive design changes requiring higher logic capacity than is available in the initially chosen device, the new design may be able to fit into a larger pin-compatible device using the same pin assignments. The same board may be used with a higher density device without the expense of board rework.



DS063_11_110501

Figure 11: Output slew-Rate for (a) Rising and (b) Falling Outputs



DS063_12_110501

Figure 12: XC9500 Devices in (a) 5V Systems and (b) Mixed 5V/3.3V Systems

In-System Programming

XC9500 devices are programmed in-system via a standard 4-pin JTAG protocol, as shown in [Figure 13](#). In-system programming offers quick and efficient design iterations and eliminates package handling. The Xilinx development system provides the programming data sequence using a Xilinx download cable, a third-party JTAG development system, JTAG-compatible board tester, or a simple microprocessor interface that emulates the JTAG instruction sequence.

All I/Os are 3-stated and pulled high by the IOB resistors during in-system programming. If a particular signal must remain Low during this time, then a pulldown resistor may be added to the pin.

External Programming

XC9500 devices can also be programmed by the Xilinx HW130 device programmer as well as third-party programmers. This provides the added flexibility of using pre-programmed devices during manufacturing, with an in-system programmable option for future enhancements.

Endurance

All XC9500 CPLDs provide a minimum endurance level of 10,000 in-system program/erase cycles. Each device meets all functional, performance, and data retention specifications within this endurance limit.

IEEE 1149.1 Boundary-Scan (JTAG)

XC9500 devices fully support IEEE 1149.1 boundary-scan (JTAG). EXTEST, SAMPLE/PRELOAD, BYPASS, USERCODE, INTEST, IDCODE, and HIGHZ instructions are supported in each device. For ISP operations, five additional instructions are added; the ISPEN, FERASE, FPGM, FVIFY, and ISPEX instructions are fully compliant extensions of the 1149.1 instruction set.

The TMS and TCK pins have dedicated pull-up resistors as specified by the IEEE 1149.1 standard.

Boundary Scan Description Language (BSDL) files for the XC9500 are included in the development system and are available on the Xilinx FTP site.

Design Security

XC9500 devices incorporate advanced data security features which fully protect the programming data against unauthorized reading or inadvertent device erasure/reprogramming. [Table 3](#) shows the four different security settings available.

The read security bits can be set by the user to prevent the internal programming pattern from being read or copied. When set, they also inhibit further program operations but allow device erase. Erasing the entire device is the only way to reset the read security bit.

The write security bits provide added protection against accidental device erasure or reprogramming when the JTAG pins are subject to noise, such as during system power-up. Once set, the write-protection may be deactivated when the device needs to be reprogrammed with a valid pattern.

Table 3: Data Security Options

		Read Security	
		Default	Set
Write Security	Default	Read Allowed Program/Erase Allowed	Read Inhibited Program Inhibited Erase Allowed
	Set	Read Allowed Program/Erase Inhibited	Read Inhibited Program/Erase Inhibited

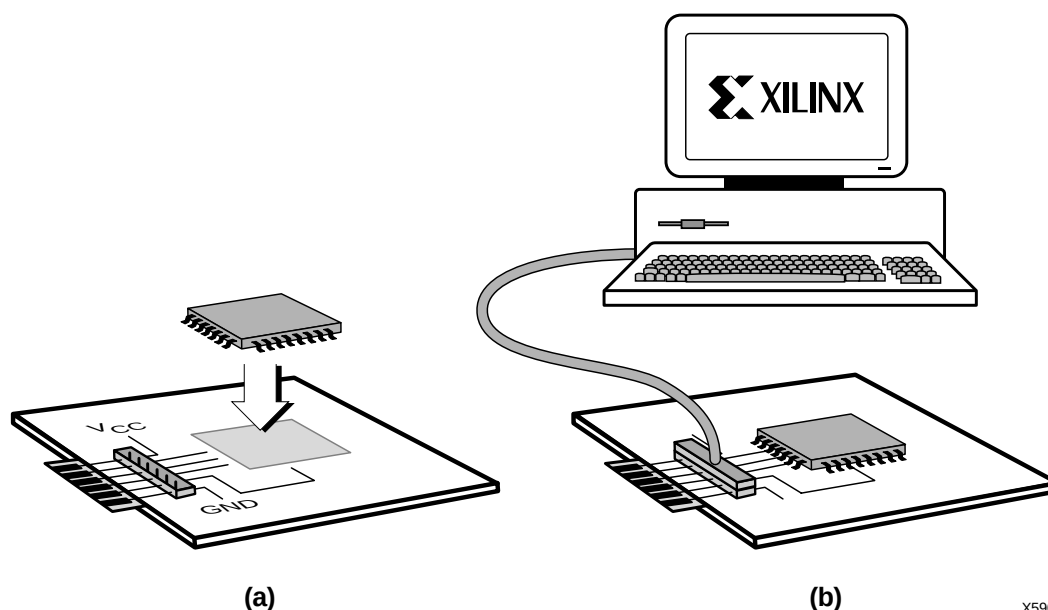


Figure 13: In-System Programming Operation (a) Solder Device to PCB and (b) Program Using Download Cable

Low Power Mode

All XC9500 devices offer a low-power mode for individual macrocells or across all macrocells. This feature allows the device power to be significantly reduced.

Each individual macrocell may be programmed in low-power mode by the user. Performance-critical parts of the application can remain in standard power mode, while other parts of the application may be programmed for low-power operation to reduce the overall power dissipation. Macrocells programmed for low-power mode incur additional delay (T_{LP}) in pin-to-pin combinatorial delay as well as register setup time. Product term clock to output and product term output enable delays are unaffected by the macrocell power-setting.

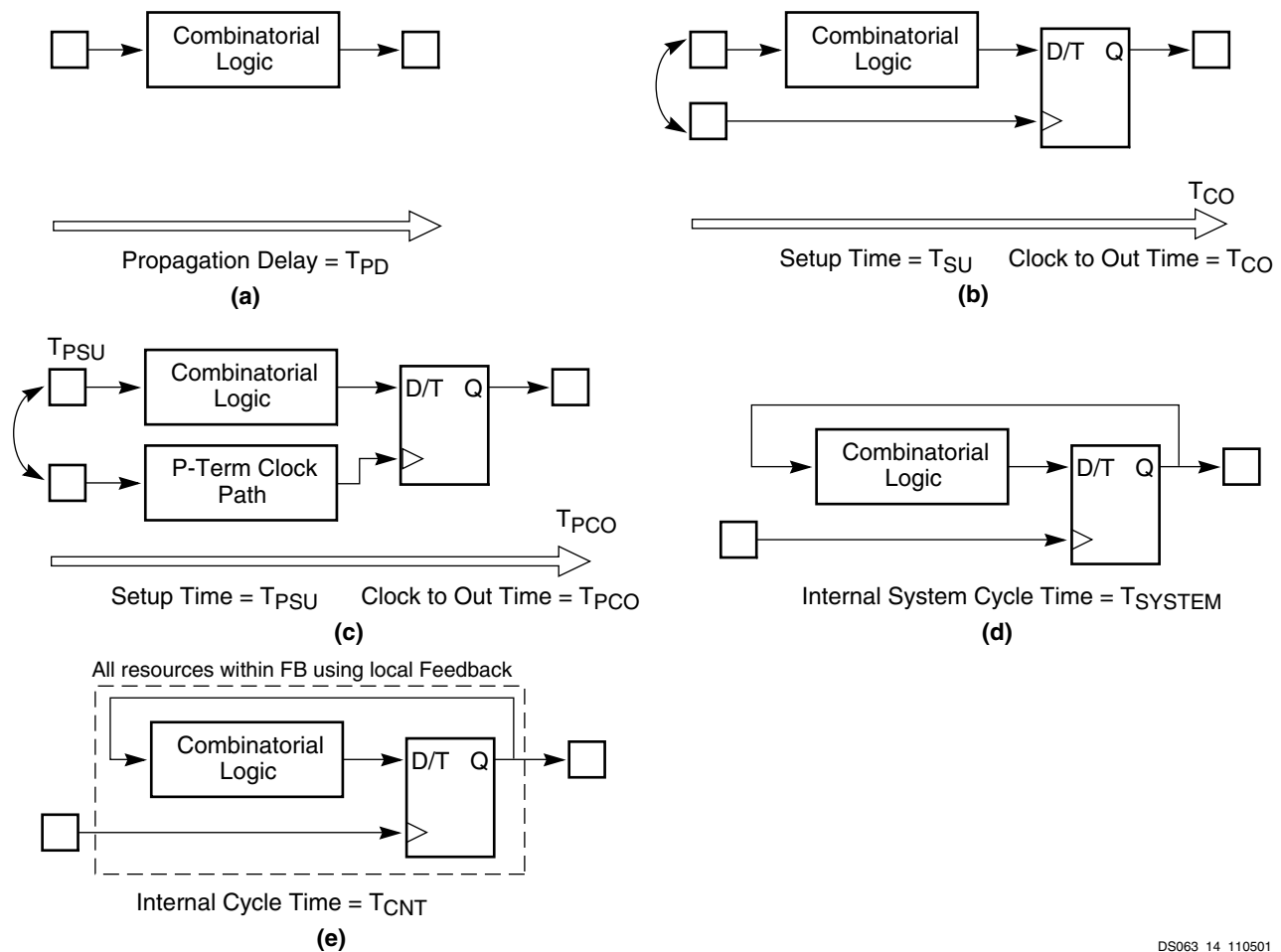
Timing Model

The uniformity of the XC9500 architecture allows a simplified timing model for the entire device. The basic timing

model, shown in Figure 14, is valid for macrocell functions that use the direct product terms only, with standard power setting, and standard slew rate setting. Table 4 shows how each of the key timing parameters is affected by the product term allocator (if needed), low-power setting, and slew-limited setting.

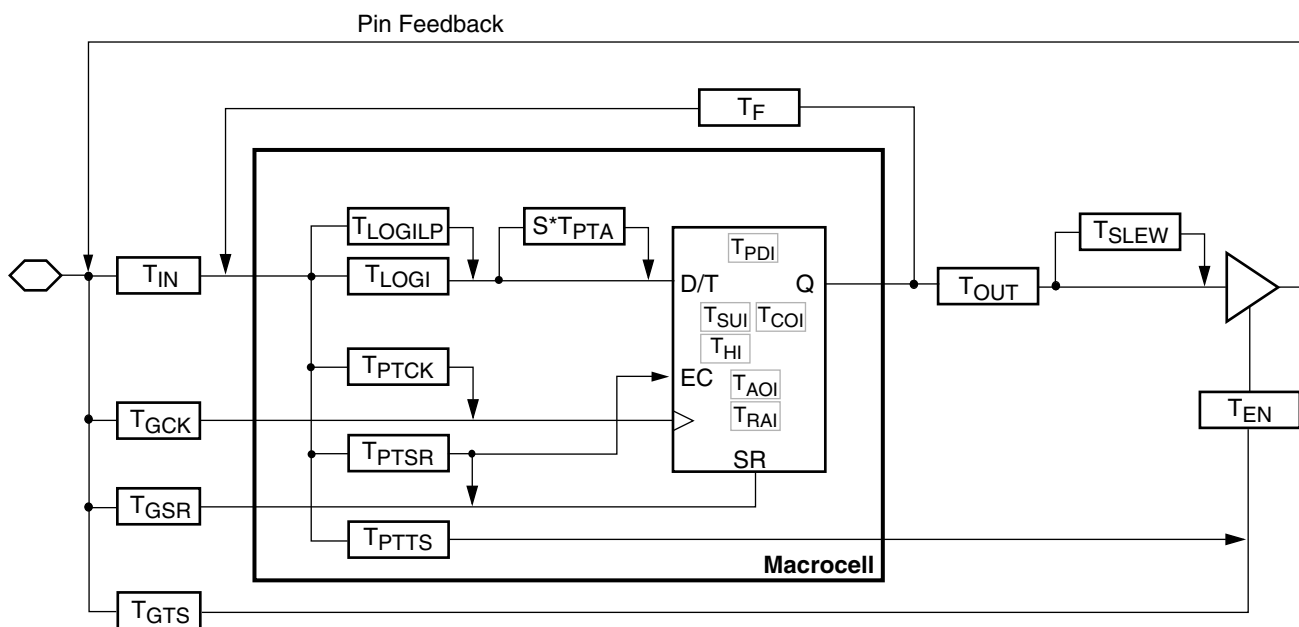
The product term allocation time depends on the logic span of the macrocell function, which is defined as one less than the maximum number of allocators in the product term path. If only direct product terms are used, then the logic span is 0. The example in Figure 6 shows that up to 15 product terms are available with a span of 1. In the case of Figure 7, the 18 product term function has a span of 2.

Detailed timing information may be derived from the full timing model shown in Figure 15. The values and explanations for each parameter are given in the individual device data sheets.



DS063_14_110501

Figure 14: Basic Timing Model



DS063_15_110501

Figure 15: Detailed Timing Model

Power-Up Characteristics

The XC9500 devices are well behaved under all operating conditions. During power-up each XC9500 device employs internal circuitry which keeps the device in the quiescent state until the V_{CCINT} supply voltage is at a safe level (approximately 3.8V). During this time, all device pins and JTAG pins are disabled and all device outputs are disabled with the IOB pull-up resistors (~10K ohms) enabled, as shown in Table 5. When the supply voltage reaches a safe level, all user registers become initialized (typically within 100 μ s for 9536, 95144, 200 μ s for 95216, and 300 μ s for 95288), and the device is immediately available for operation, as shown in Figure 16.

If the device is in the erased state (before any user pattern is programmed), the device outputs remain disabled with the IOB pull-up resistors enabled. The JTAG pins are enabled to allow the device to be programmed at any time.

If the device is programmed, the device inputs and outputs take on their configured states for normal operation. The JTAG pins are enabled to allow device erasure or boundary-scan tests at any time.

Development System Support

The XC9500 CPLD family is fully supported by the development systems available from Xilinx and the Xilinx Alliance Program vendors.

The designer can create the design using ABEL, schematics, equations, VHDL, or Verilog in a variety of software front-end tools. The development system can be used to

implement the design and generate a JEDEC bitmap which can be used to program the XC9500 device. Each development system includes JTAG download software that can be used to program the devices via the standard JTAG interface and a download cable.

FastFLASH Technology

An advanced CMOS Flash process is used to fabricate all XC9500 devices. Specifically developed for Xilinx in-system programmable CPLDs, the FastFLASH process provides high performance logic capability, fast programming times, and endurance of 10,000 program/erase cycles.

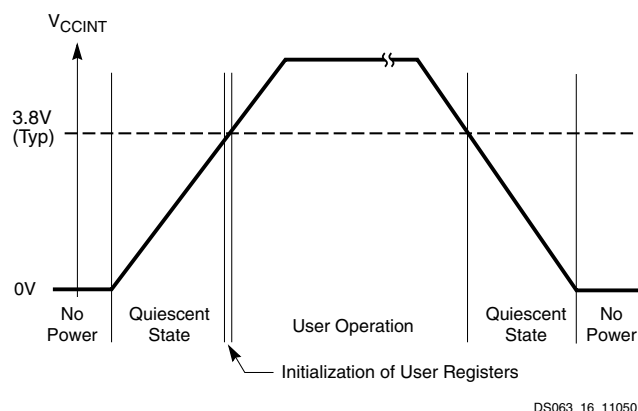


Figure 16: Device Behavior During Power-up

Table 4: Timing Model Parameters

Parameter	Description	Product Term Allocator ⁽¹⁾	Macrocell Low-Power Setting	Output Slew-Limited Setting
T_{PD}	Propagation Delay	$+ T_{PTA} * S$	$+ T_{LP}$	$+ T_{SLEW}$
T_{SU}	Global Clock Setup Time	$+ T_{PTA} * S$	$+ T_{LP}$	-
T_{CO}	Global Clock-to-output	-	-	$+ T_{SLEW}$
T_{PSU}	Product Term Clock Setup Time	$+ T_{PTA} * S$	$+ T_{LP}$	-
T_{PCO}	Product Term Clock-to-output	-	-	$+ T_{SLEW}$
T_{SYSTEM}	Internal System Cycle Period	$+ T_{PTA} * S$	$+ T_{LP}$	-

Notes:

1. S = the logic span of the function, as defined in the text.

Table 5: XC9500 Device Characteristics

Device Circuitry	Quiescent State	Erased Device Operation	Valid User Operation
IOB Pull-up Resistors	Enabled	Enabled	Disabled
Device Outputs	Disabled	Disabled	As Configured

Table 5: XC9500 Device Characteristics

Device Circuitry	Quiescent State	Erased Device Operation	Valid User Operation
Device Inputs and Clocks	Disabled	Disabled	As Configured
Function Block	Disabled	Disabled	As Configured
JTAG Controller	Disabled	Enabled	Enabled

Warranty Disclaimer

THESE PRODUCTS ARE SUBJECT TO THE TERMS OF THE XILINX LIMITED WARRANTY WHICH CAN BE VIEWED AT <http://www.xilinx.com/warranty.htm>. THIS LIMITED WARRANTY DOES NOT EXTEND TO ANY USE OF THE PRODUCTS IN AN APPLICATION OR ENVIRONMENT THAT IS NOT WITHIN THE SPECIFICATIONS STATED ON THE THEN-CURRENT XILINX DATA SHEET FOR THE PRODUCTS. PRODUCTS ARE NOT DESIGNED TO BE FAIL-SAFE AND ARE NOT WARRANTED FOR USE IN APPLICATIONS THAT POSE A RISK OF PHYSICAL HARM OR LOSS OF LIFE. USE OF PRODUCTS IN SUCH APPLICATIONS IS FULLY AT THE RISK OF CUSTOMER SUBJECT TO APPLICABLE LAWS AND REGULATIONS.

Additional Information

[XC9500 Data sheets and application notes.](#)

[Packages](#)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
3.0	12/14/98	Revised datasheet to reflect new AC characteristics and Internal Timing Parameters.
4.0	02/10/99	Corrected Figure 3.
5.0	09/15/99	Added -10 speed grade to XC95288.
5.1	09/22/03	Minor edits.
5.2	02/16/04	Corrected statement on GTS inputs on page 10. Added links to additional information.
5.3	04/15/05	Update to PDF attributes only. No changes to documentation.
5.4	04/03/06	Added Warranty Disclaimer. Added note on Pb-Free packages.
5.5	06/25/07	Change to Table 2, listing discontinuance of 352-pin BGA package for the XC95216. See XCN07010 for details.

REVISIONS			
REV.	ECO. NO	DESCRIPTION	DATE
A	1682	INITIAL RELEASE	5/8/2008
B	1852	UPDATE DIM 'C', TABLE, NOTE 3 & TOLERANCE BLOCK	11/12/08

PART NUMBER CODING

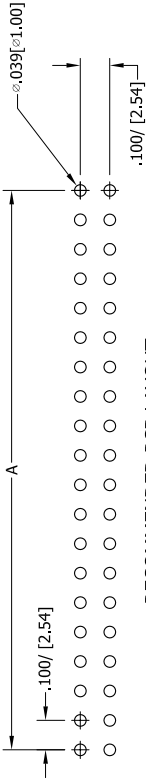
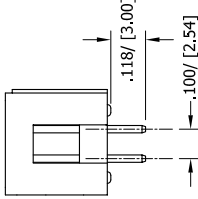
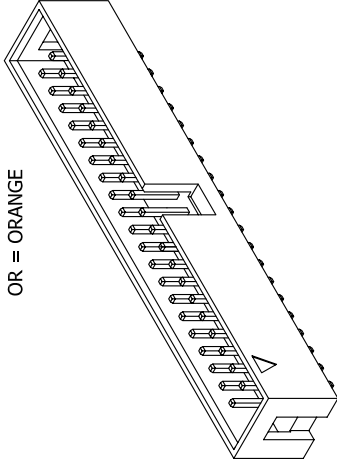
SBH11 - PBPC - D _ _ - ST - _ _

NUMBER OF POSITIONS
(CONTACTS PER ROW)

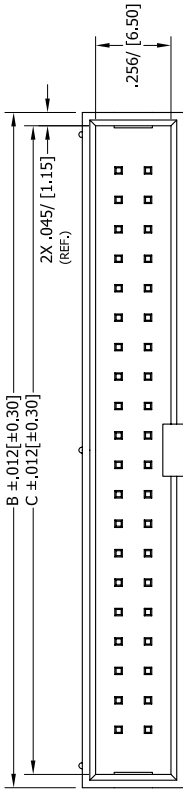
TERMINATION
STRAIGHT

COLOR

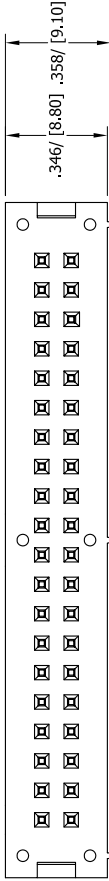
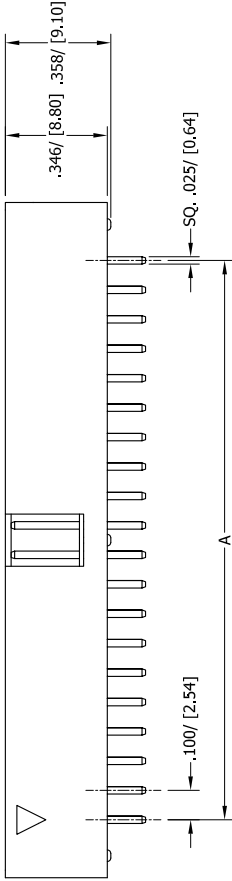
BK = BLACK
YE = YELLOW
RE = RED
BL = BLUE
GR = GREEN
GY = GREY
WH = WHITE
OR = ORANGE



RECOMMENDED PCB LAYOUT



(0.098 [2.50] FOR POSITION # 3)



PART NUMBER	POSITIONS/ CONTACTS	A		B		C	
		INCH	MM	INCH	MM	INCH	MM
SBH11-PBPC-D03-ST-__	03/06	0.200	5.08	0.595	15.12	0.505	12.82
SBH11-PBPC-D04-ST-__	04/08	0.300	7.62	0.695	17.66	0.605	15.36
SBH11-PBPC-D05-ST-__	05/10	0.400	10.16	0.795	20.20	0.705	17.90
SBH11-PBPC-D06-ST-__	06/12	0.500	12.70	0.895	22.74	0.805	20.44
SBH11-PBPC-D07-ST-__	07/14	0.600	15.24	0.995	25.28	0.905	22.98
SBH11-PBPC-D08-ST-__	08/16	0.700	17.78	1.095	27.82	1.005	25.52
SBH11-PBPC-D10-ST-__	10/20	0.900	22.86	1.295	32.90	1.205	30.60
SBH11-PBPC-D12-ST-__	12/24	1.100	27.94	1.495	37.98	1.405	35.68
SBH11-PBPC-D13-ST-__	13/26	1.200	30.48	1.595	40.52	1.505	38.22
SBH11-PBPC-D15-ST-__	15/30	1.400	35.56	1.795	45.60	1.705	43.30
SBH11-PBPC-D17-ST-__	17/34	1.600	40.64	1.995	50.68	1.905	48.38
SBH11-PBPC-D20-ST-__	20/40	1.900	48.26	2.295	58.30	2.205	56.00
SBH11-PBPC-D22-ST-__	22/44	2.100	53.34	2.495	63.38	2.405	61.08
SBH11-PBPC-D25-ST-__	25/50	2.400	60.96	2.795	71.00	2.705	68.70
SBH11-PBPC-D30-ST-__	30/60	2.900	73.66	3.295	83.70	3.205	81.40
SBH11-PBPC-D32-ST-__	32/64	3.100	78.74	3.495	88.78	3.405	86.48

NOTES:

- INSULATOR MATERIAL: PBT.
- CONTACT MATERIAL: BRASS.
- PLATING: GOLD FLASH OVERALL.
- OPERATING TEMPERATURE: -40°C ~ 85°C.
- PROCESSING TEMP: 230°C FOR 60 SECONDS MAX.
(260°C, 10 SEC.)
- UL FLAMMABILITY RATING: 94V-0.
- VOLTAGE RATING: 20m OHMS MAX.
- CURRENT RATING: 3 AMP PER CONTACT.
- VOLTAGE DROP: AC 500 V.
- INSULATION RESISTANCE: 1000 MEGA OHMS.



RoHS COMPLIANT

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES (MM)

TOLERANCES:
ANGULAR: ± 1°
XX± = ±.012 [0.30]
XXX± = ±.008 [0.20]
XXXX± = ±.004 [0.10]
PARENT DIMENSIONAL INFORMATION IS FOR
REFERENCE ONLY

INTERPRET DIMENSIONS AND GEOMETRIC
TOLERANCING
PER: ANSI Y14.5-1994

DRAWN: V. JARA
DATE: 5/8/2008

THESE DIMENSIONS ARE FOR INFORMATION ONLY
SULLINS ELECTRONICS AND ITS NET
WORKS DO NOT ASSUME ANY
DISCLOSED TO OTHERS FOR ANY
PURPOSE WITHOUT THE WRITTEN
AUTHORIZATION OF SULLINS ELECTRONICS.

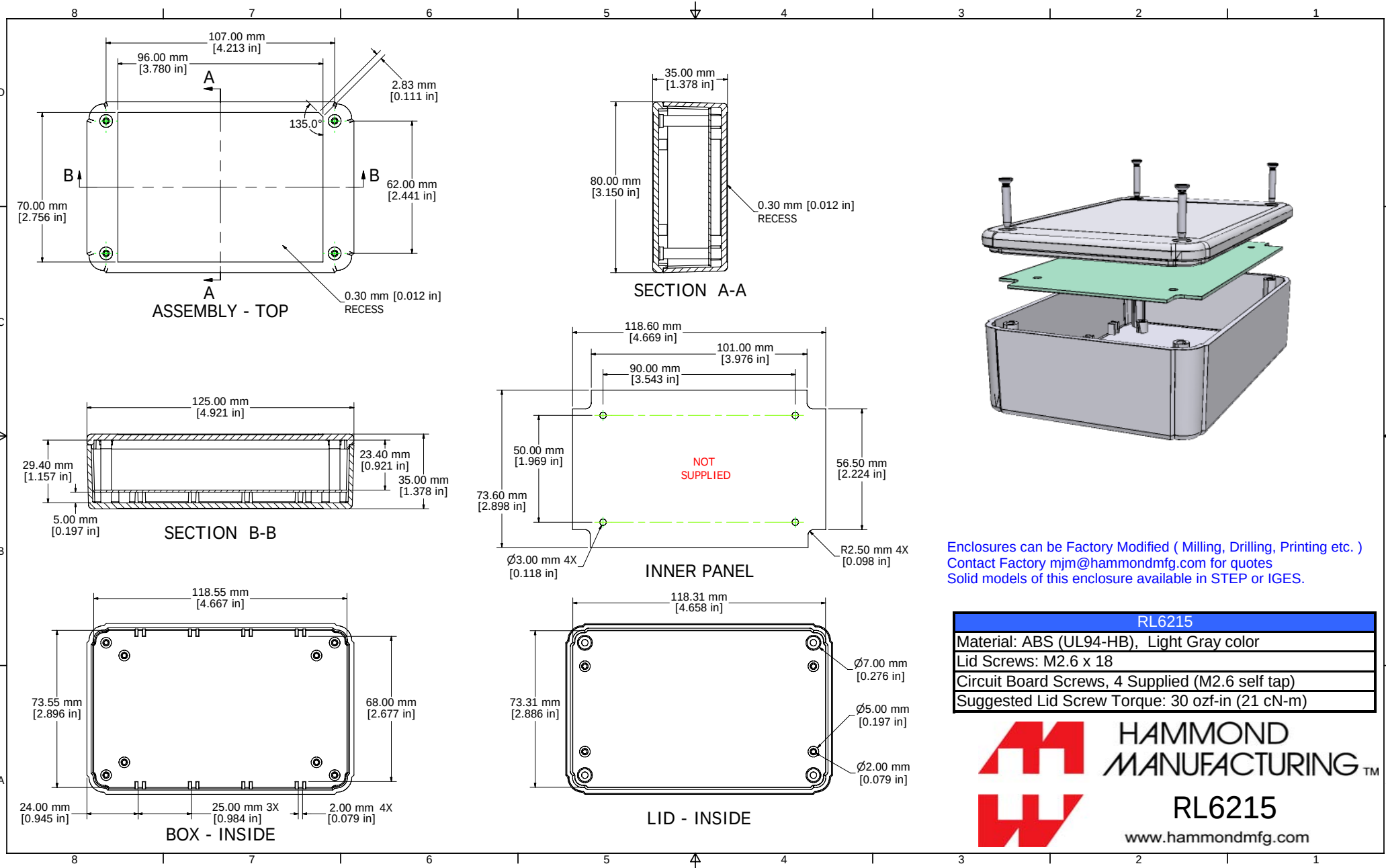
SULLINS
CONNECTOR SOLUTIONS

DESIGNATION: SBH11-PBPC-D-ST-
PART NUMBER: SBH11-PBPC-D-ST-
SIZE (DWG. NO.): C 11091

REV: B

SCALE: NONE

SHEET 1 OF 1



Enclosures can be Factory Modified (Milling, Drilling, Printing etc.)
Contact Factory mjm@hammondmg.com for quotes
Solid models of this enclosure available in STEP or IGES.

RL6215
Material: ABS (UL94-HB), Light Gray color
Lid Screws: M2.6 x 18
Circuit Board Screws, 4 Supplied (M2.6 self tap)
Suggested Lid Screw Torque: 30 ozf-in (21 cN-m)



**HAMMOND
MANUFACTURING™**

RL6215

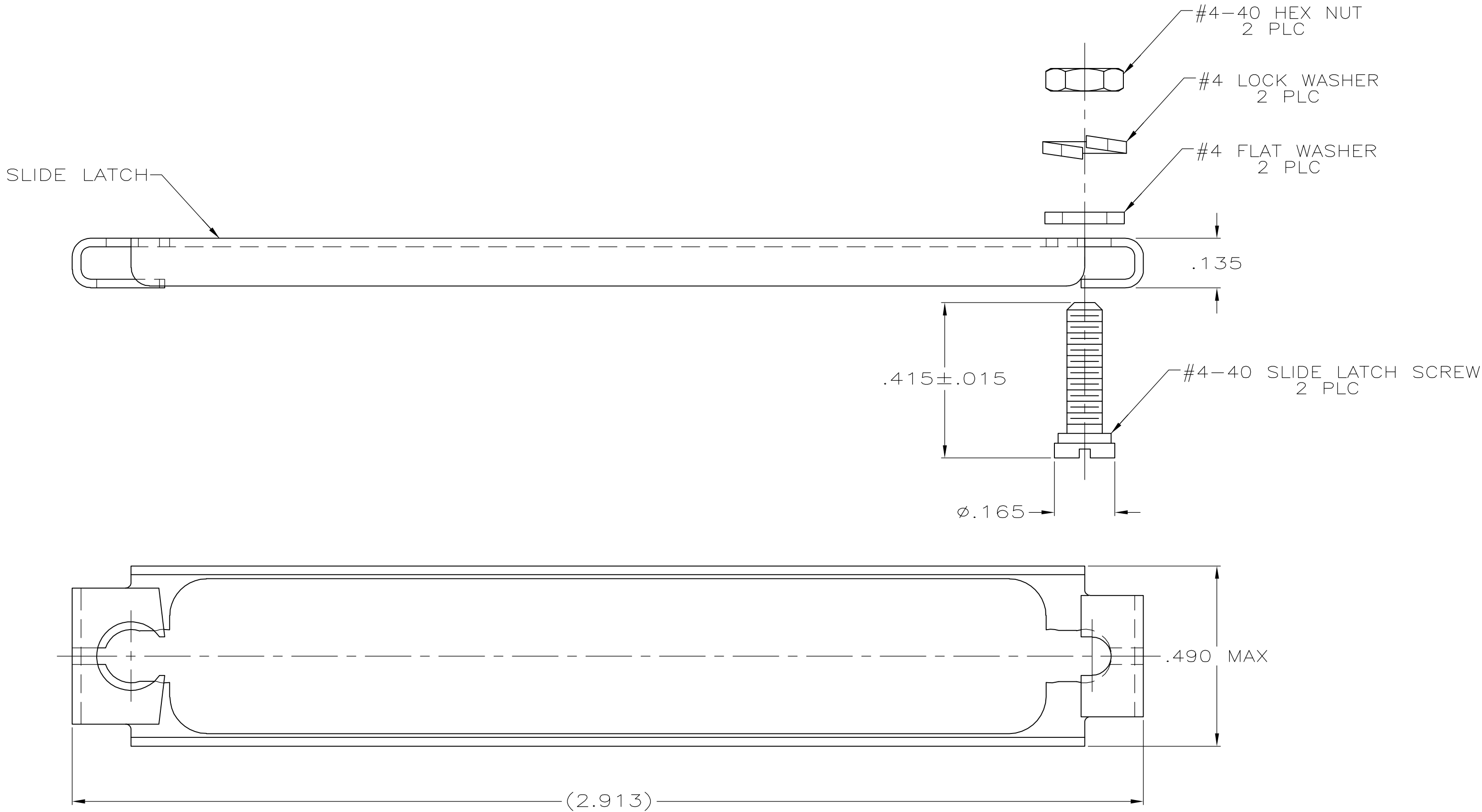
www.hammondmg.com

THIS DRAWING IS UNPUBLISHED.

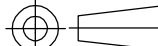
RELEASED FOR PUBLICATION

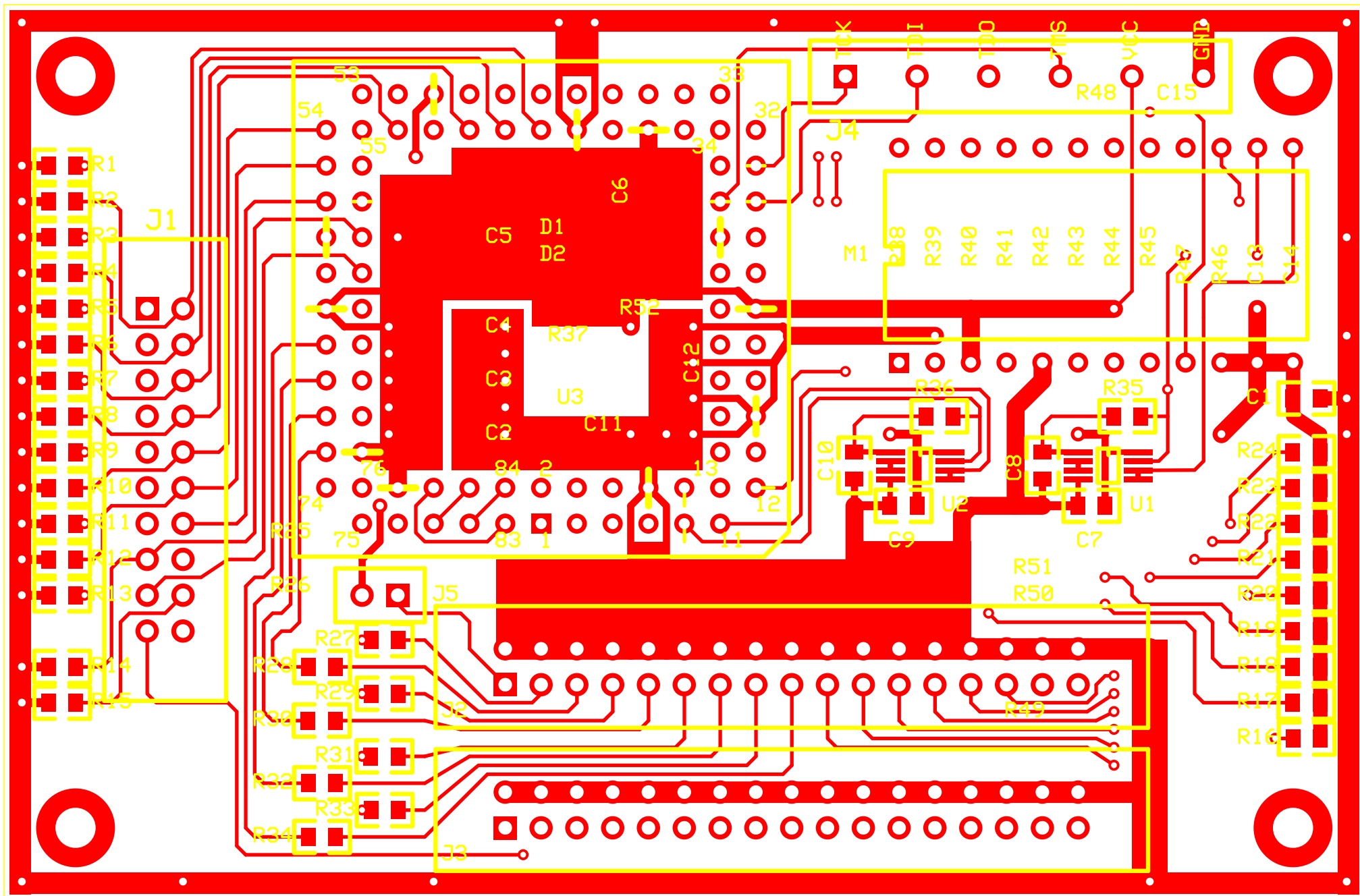
© COPYRIGHT - BY TYCO ELECTRONICS CORPORATION. ALL RIGHTS RESERVED.

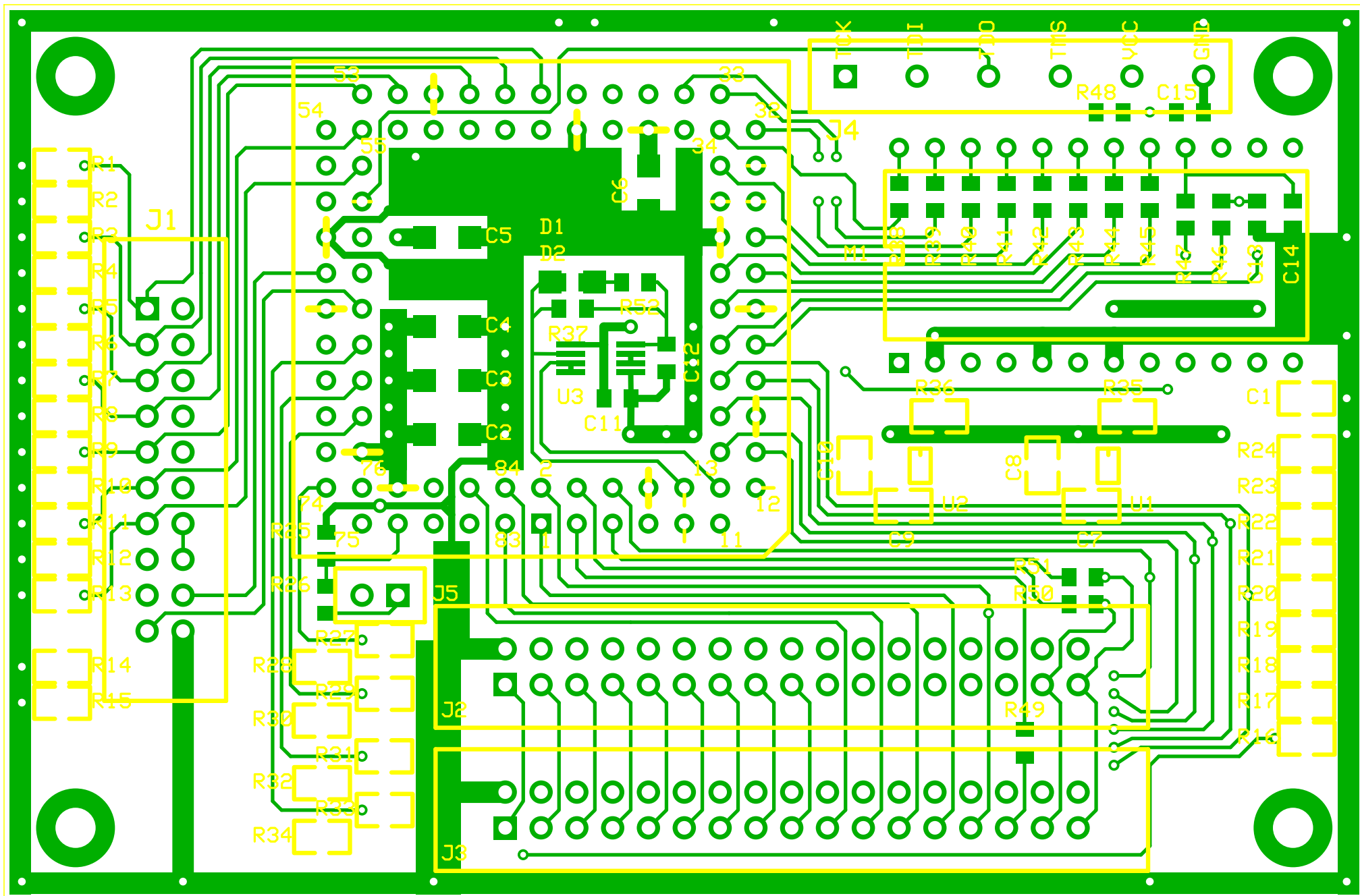
LOC		DIST		REVISIONS			
P	LTR	DESCRIPTION		DATE	DWN	APVD	
	0	RELEASED PER EC 0S13-0207-05		29JUN05	SW	JG	

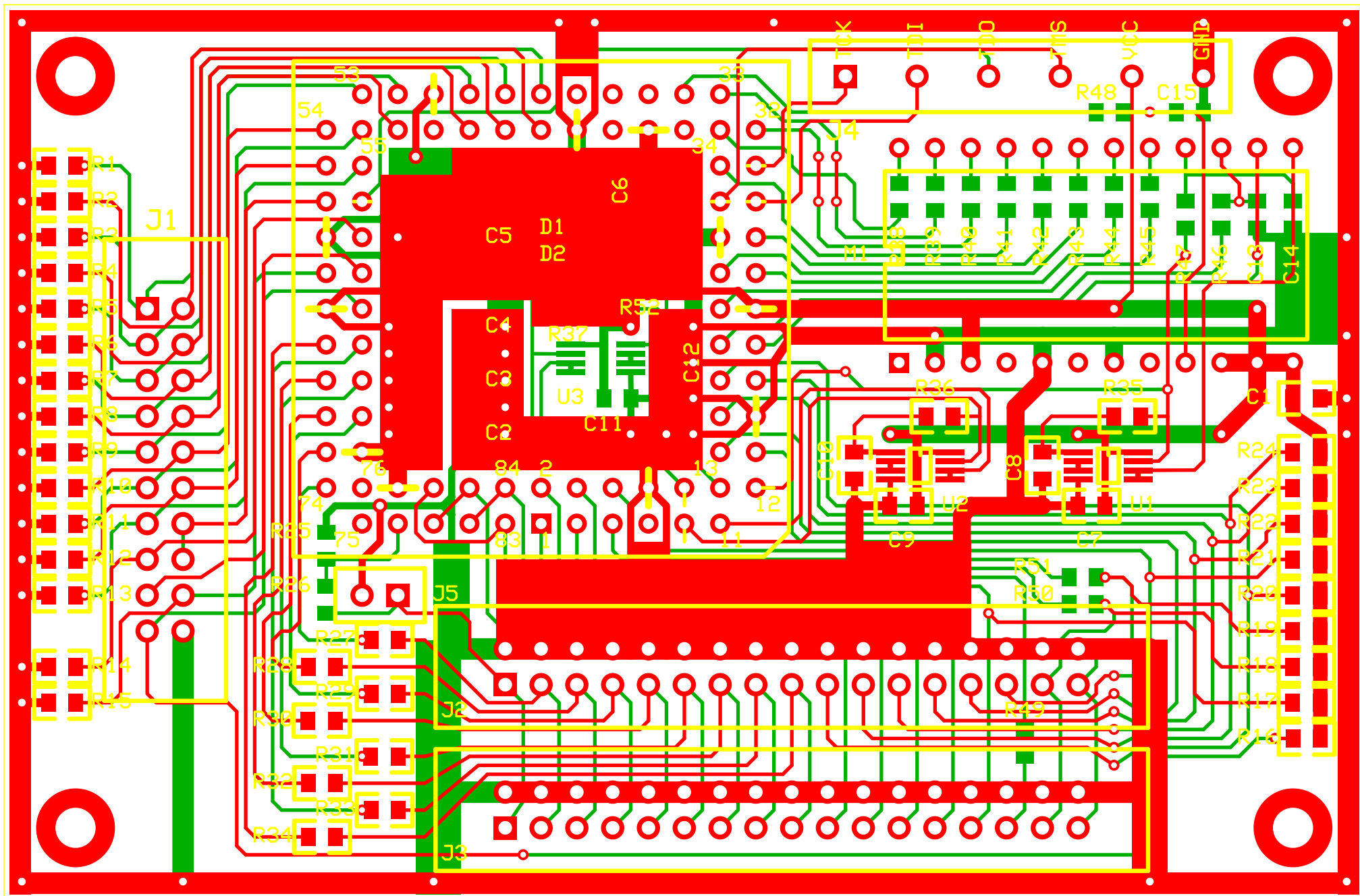


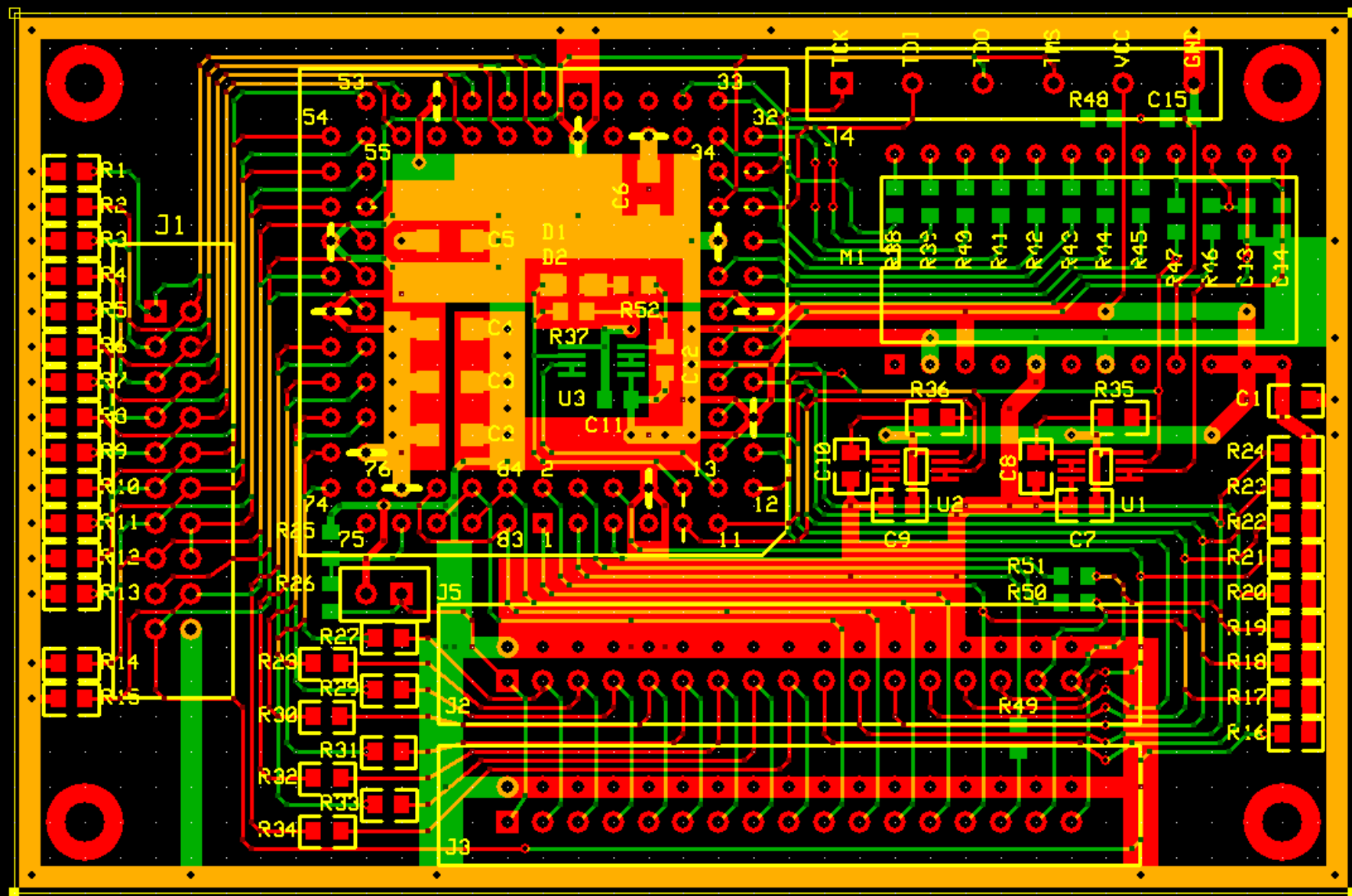
- 1 . PARTS PACKAGED UNASSEMBLED.
2. PARTS PACKAGED AS A UNIT KIT.
3. THIS KIT CAN BE USED WITH CABLE CLAMP BODIES THAT DO NOT EXTEND BEYOND THE FRONT FLANGE OF THE CONNECTOR OR ON CONNECTORS WITHOUT CABLE CLAMPS.
- 4 ZINC PLATED .000200-.000400 THICK, YELLOW CHROMATE COATING.

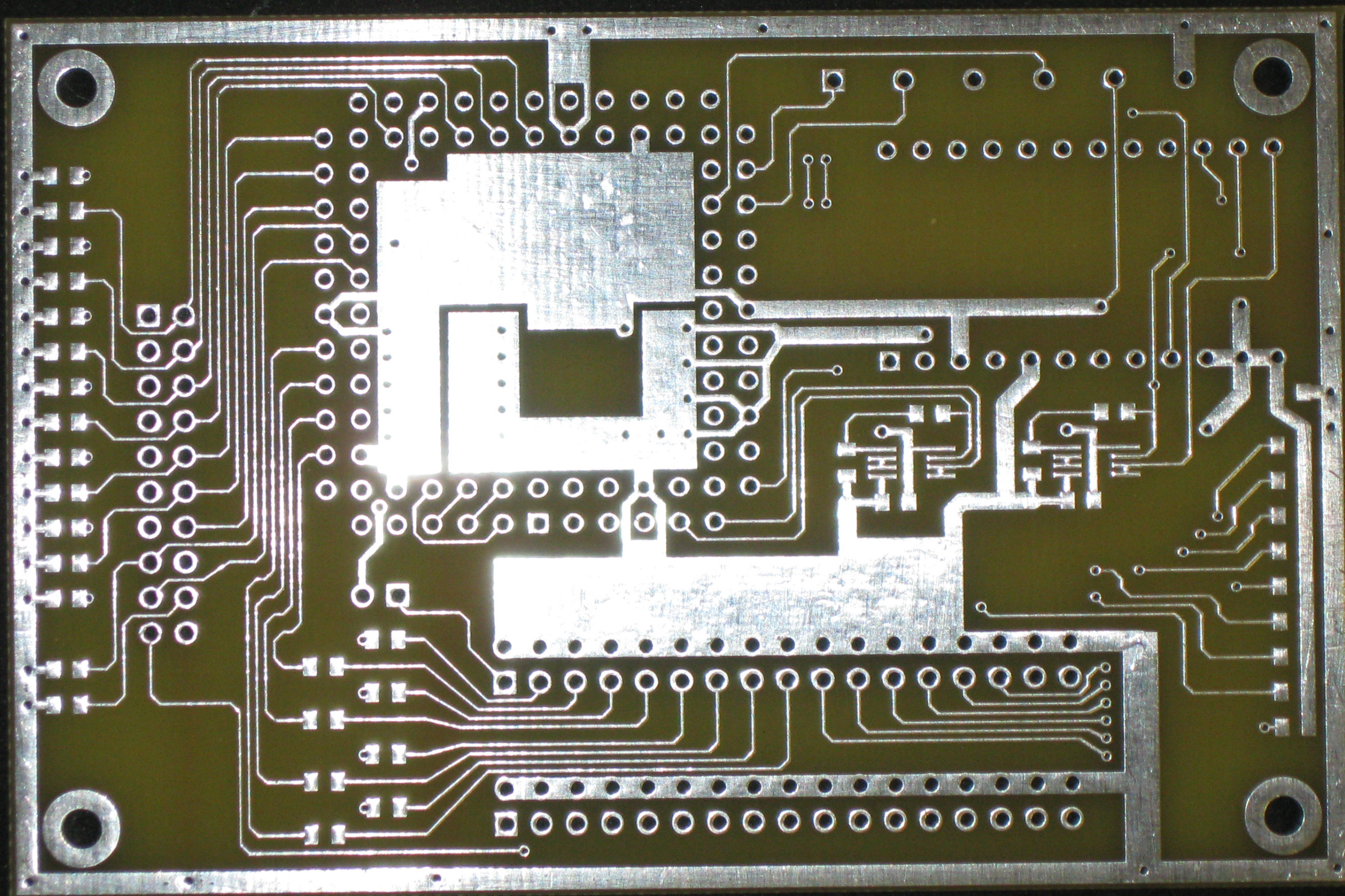
THIS DRAWING IS A CONTROLLED DOCUMENT.		DWN S.WAITMAN	29JUN05	tyco Electronics Tyco Electronics Corporation Harrisburg, Pa 17105-3608			
		CHK C.ROHDE	29JUN05				
DIMENSIONS: INCHES		APVD M.WALMSLEY	29JUN05	NAME KIT, SLIDE LATCH, SIZE 4, 37 POSITION, AMPLIMITE —			
		PRODUCT SPEC —					
		APPLICATION SPEC —	SIZE A2		CAGE CODE 00779	DRAWING NO C=5745407	RESTRICTED TO —
MATERIAL SLIDE LATCH: STAINLESS STEEL HARDWARE: STEEL		FINISH HARDWARE: 	WEIGHT —				
CUSTOMER DRAWING			SCALE 4:1		SHEET 1 OF 1	REV 0	

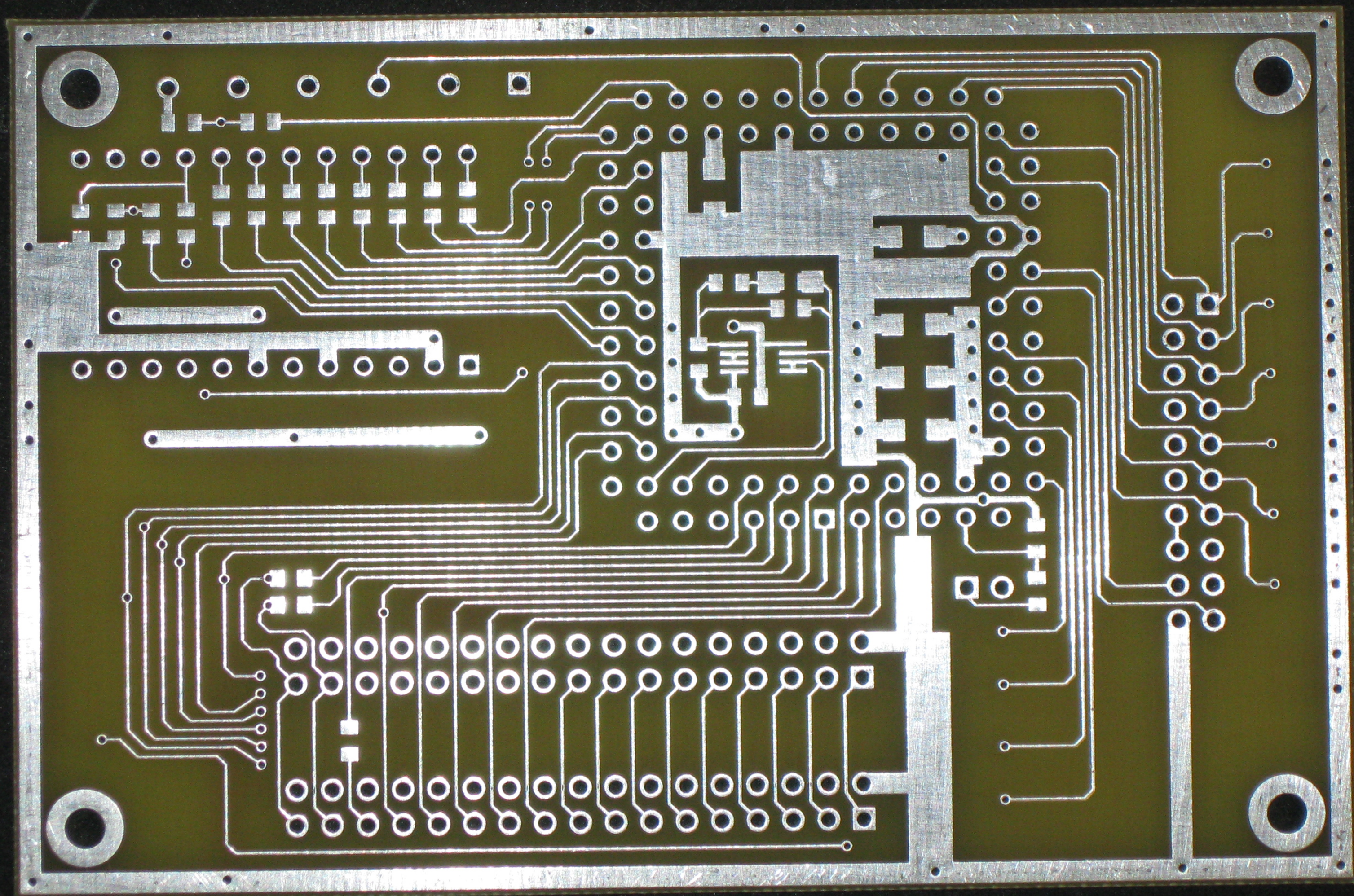


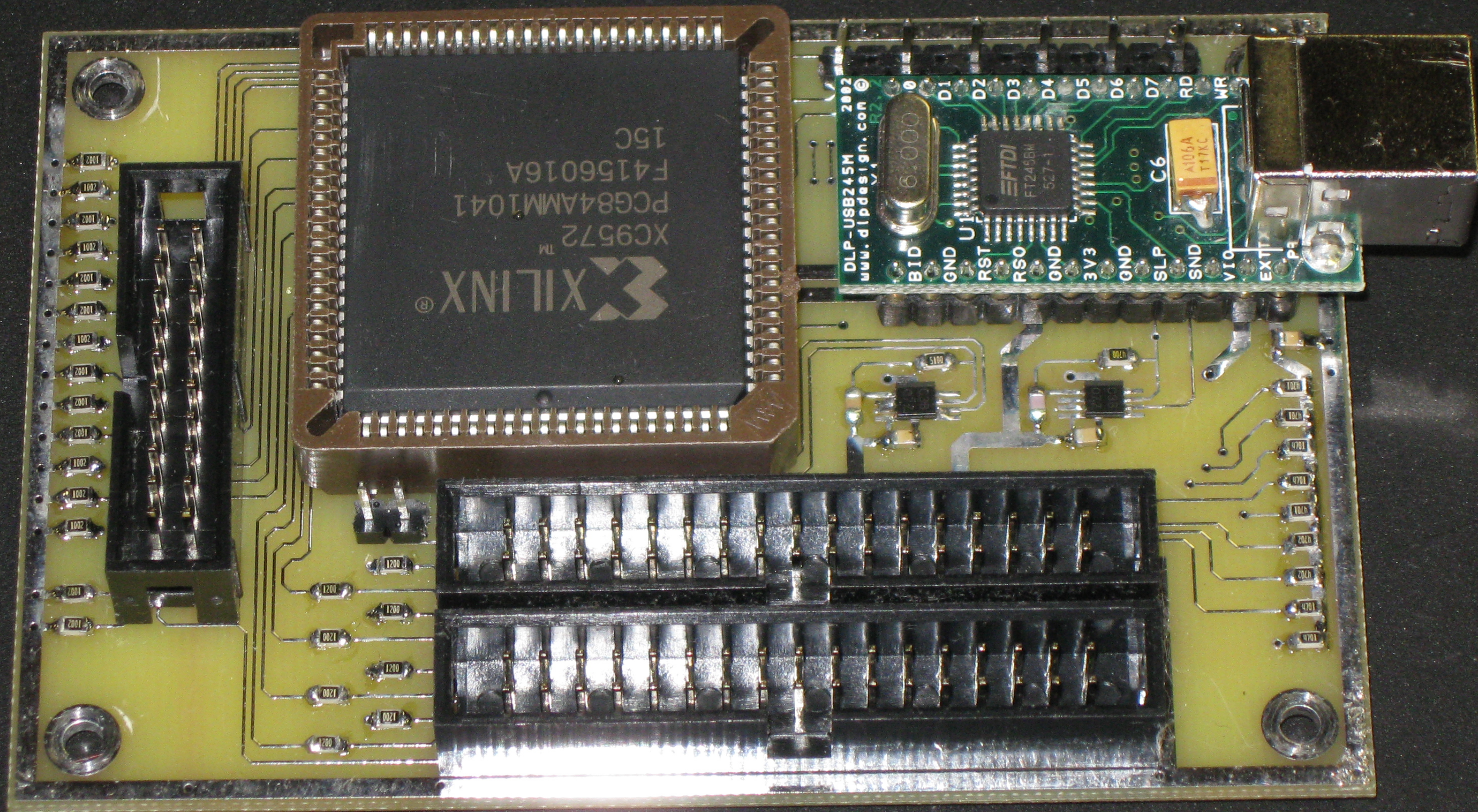


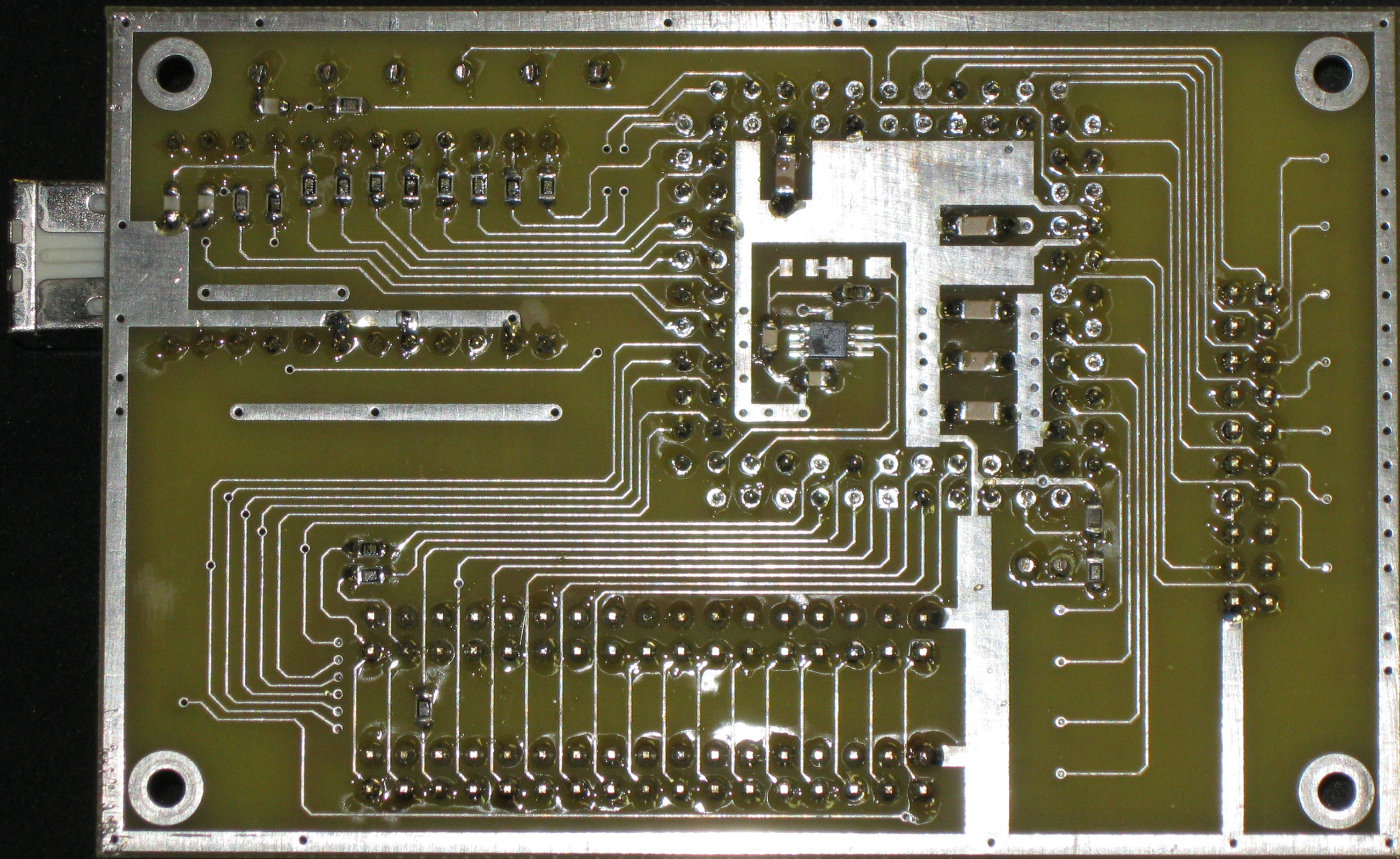


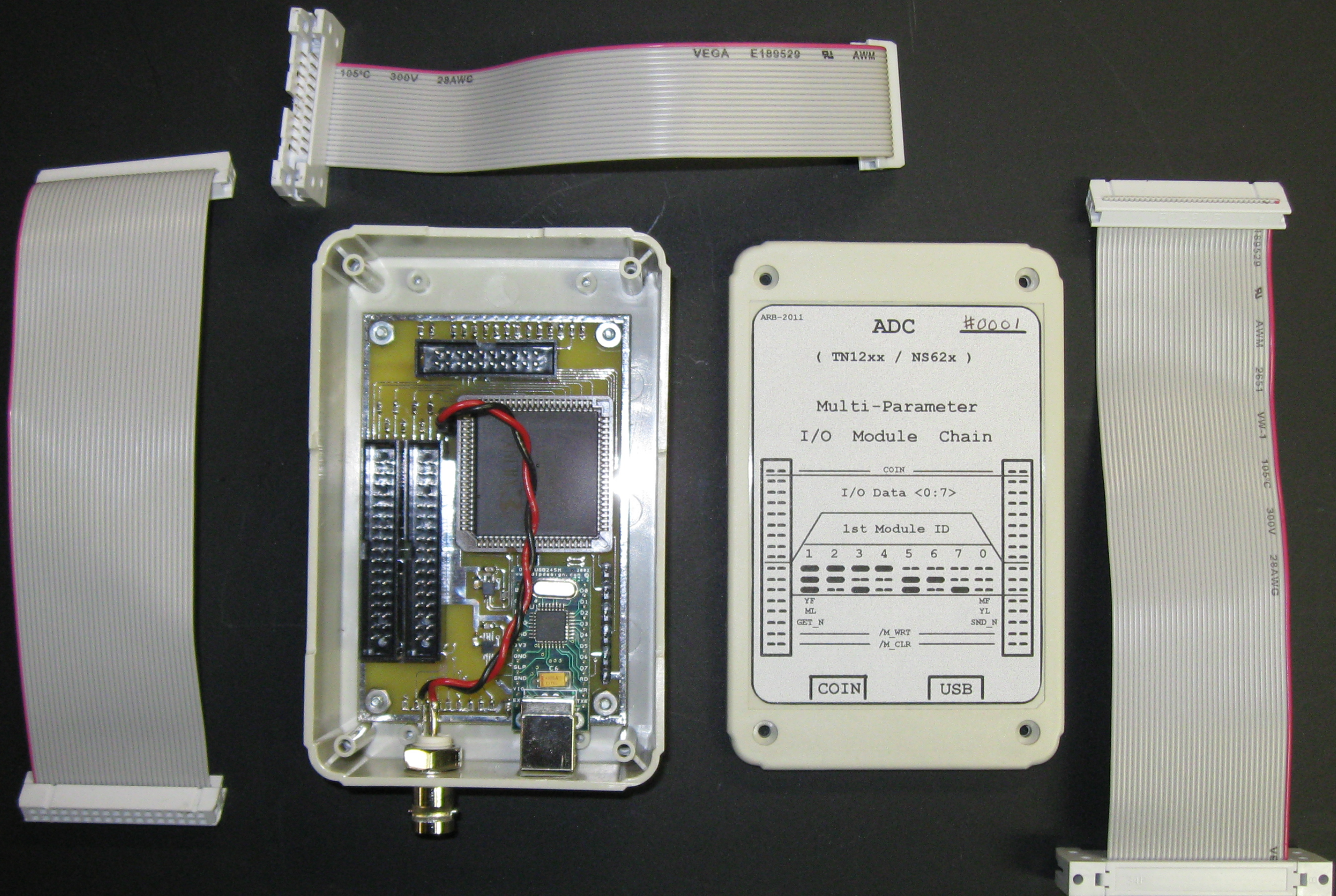


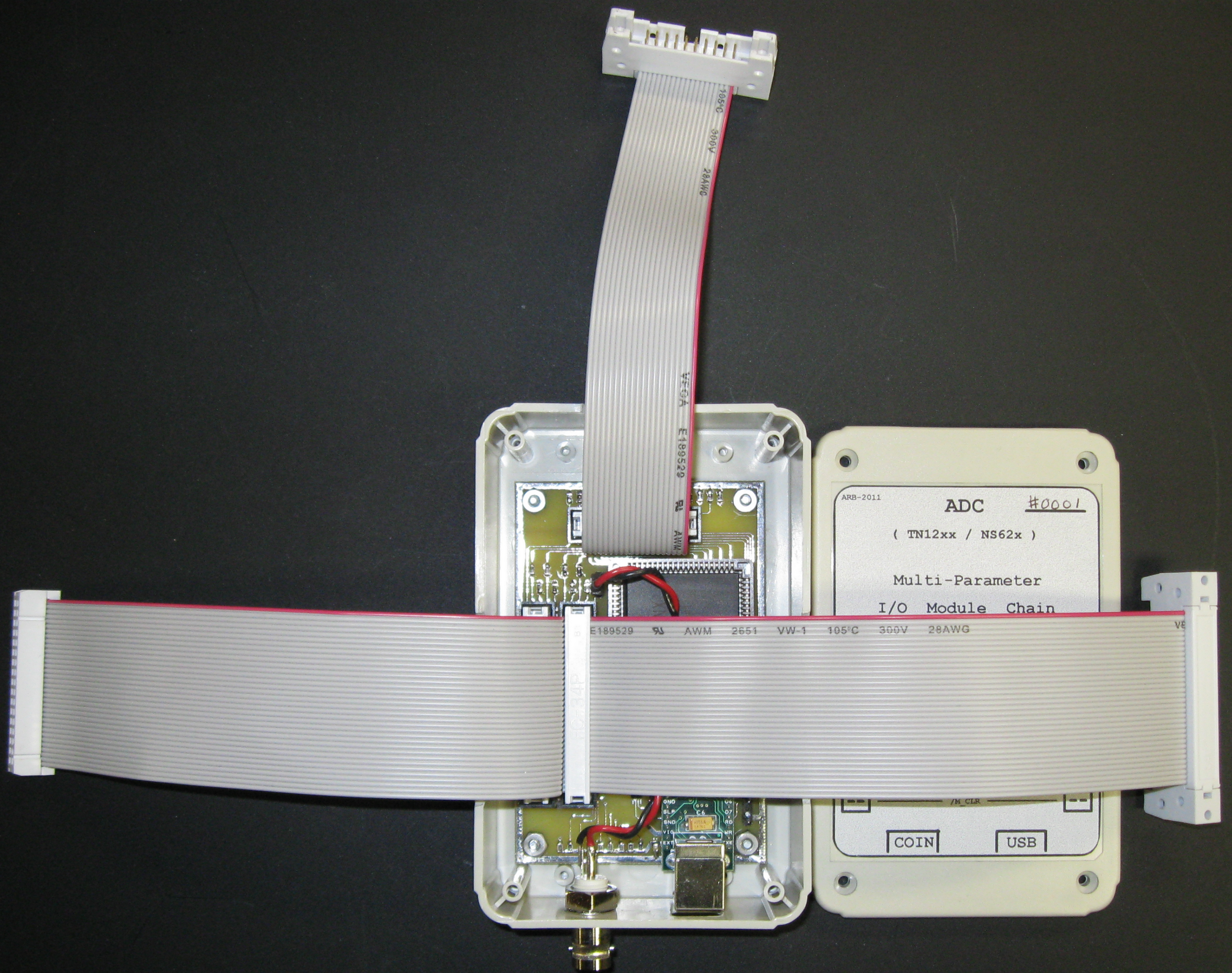












ARB-2011

ADC

#0001

(TN12xx / NS62x)

Multi-Parameter
I/O Module Chain

E189529 AWM 2651 VW-1 105°C 300V 28AWG

COIN

USB



ARB-2011

ADC

#00001

(TN12xx / NS62x)

Multi-Parameter
I/O Module Chain

COIN

I/O Data <0:7>

1st Module ID

1	2	3	4	5	6	7	0
							
YF	YF	YF	YF	YF	YF	YF	YF
ML	ML	ML	ML	ML	ML	ML	ML
GET_N	GET_N	GET_N	GET_N	GET_N	GET_N	GET_N	GET_N
				/M_WRT			
				/M_CLR			

