

1.0 Introduction

USB2Any



USB 2.0 to I2C/SPI/GPIO/SPP/EPP/ADC/RS232(TTL)/JTAG adapter

USB2Any provides a simple solution to control various hardware devices with I2C, SPI, RS232(TTL), JTAG, ADC, SPP/EPP, GPIO interfaces from your PC. **Multiple commands** can be easily packed into one USB transaction frame(which takes 1 ms minimum for low and full speed device), instead of one command per transaction as other similar product does, so tremendous execution efficiency can be achieved by USB2Any.

1.1 Features:

- In a compact, rigid plastic cover with mini-B to DB25 interface connector.
- Provide a 3.3V/180mA power source to support user applications.
- Configurable I2C, SPI and RS232(TTL) bus clock frequency.
- One I2C master port, one SPI master port, one RS232 port in TTL level, one SPP/EPP parallel port, one JTAG port and two single-end analog input channels.
- Up to 23 user configurable pins which, through the ability of writing the SFR registers the USB2Any provided, can accomplish almost any function the C8051F320 can have, such as a 10 channel differential analog inputs AD converter with internal voltage reference.
- Flexible and powerful DLL software interface and plenty of examples with source files on read/write a LCD module, a 25 series flash device, a 24 series EEPROM and a CPLD JTAG interface.
- User can build his own device commands to fit his hardware so that his application can run efficiently.

1.2 Device Command summary:

// General

CmdGetVer, CmdPowerOn, CmdPowerOff, CmdSFRRead, CmdSFRWrite,
CmdSetPWidth, CmdWaitN100uS, CmdSelEPP, CmdSelSPP, CmdSelSIO, CmdLoadReportN,

// SPI

CmdSPI0SetRate, CmdSPI0En, CmdSPI0Dis, CmdSPI0CS0_Byte, CmdSPI0CS1, CmdSPI0WriteN,
CmdSPI0ReadN, CmdWaitWIPL, CmdSPI0UnloadN, CmdSPI0ReadReportL,

// I2C

CmdSMB0SetRate, CmdSMB0En, CmdSMB0Dis, CmdSMB0Start, CmdSMB0Stop, CmdSMB0NACKStop,
CmdSMB0ReadN, CmdSMB0Read_NACK, CmdSMB0WriteN, CmdSMB0UnloadN,
CmdSMB0ReadReportL,

// SPP/EPP PORT I/O

CmdPBitSet, CmdPBitClr, CmdStbWrite, CmdStbWriteN, CmdDStbWrite, CmdDStbRead, CmdAStbWrite,
CmdAStbRead, CmdWaitBit0, CmdWaitBit1,

// JTAG

CmdJTAGReset, CmdTCK_TMS1, CmdTCK_TMS0, CmdTDI1, CmdTDI0, CmdTCKsL, CmdBitsRWExitL,
CmdBitsRWL,

// ADC

CmdADC0En, CmdADC0Dis, CmdADC0SetAMX, CmdADC0Get,

// RS232(TTL)

Cmd232SetBaud, Cmd232Init, Cmd232Read_Flag, Cmd232ReadN, Cmd232WriteN,

// Power User

CmdUserReadPage, CmdUserWritePage, CmdUser0, CmdUser1, CmdUser2, CmdUser3, CmdUser4,
CmdUser5, CmdUser6, CmdUser7, CmdUser8, CmdUser9

2.0 USB2Any pin assignment:

	SPP/EPP mode	SIO mode Note7	
P0.0(DB25.21)	ODI	SCK(SPI)	Note4
P0.1(DB25.22)	ODI	MISO(SPI)	Note4
P0.2(DB25.23)	ODI	MOSI(SPI)	Note4
P0.3(DB25.15)	nERROR, ODI	NSS(SPI)	
P0.4(DB25.24)	SEL, ODI	TX(RS232)	Note4
P0.5(DB25.12)	PE, ODI	RX(RS232)	
P0.6(DB25.10)	nACK, ODI	SDA(I2C)	Note6
P0.7(DB25.11)	BUSY/WAIT, ODI	SCL(I2C)	Note5,6
P1.0(DB25.2)	PD0, PP/ODI	ODI	
P1.1(DB25.3)	PD1, PP/ODI	ODI	
P1.2(DB25.4)	PD2, PP/ODI	ODI	
P1.3(DB25.5)	PD3, PP/ODI	ODI	
P1.4(DB25.6)	PD4, PP/ODI	ODI	
P1.5(DB25.7)	PD5, PP/ODI	ODI	
P1.6(DB25.8)	PD6, PP/ODI	ODI	
P1.7(DB25.9)	PD7, PP/ODI	ODI	
P2.0(DB25.1)	nSTB/nWrite, TCK(JTAG), PP		Note5
P2.1(DB25.14)	nAF/nDSTB, TDI(JTAG), PP		Note5
P2.2(DB25.16)	nINIT/nReset, TMS(JTAG), PP		
P2.3(DB25.17)	nSELIN/nASTB, TRST(JTAG), PP		Note5
P2.4(DB25.18)	ODI, TDO(JTAG)		Note4
P2.5(DB25.19)	ODI, AIN0(ADC)		Note4
P2.6(DB25.20)	ODI, AIN1(ADC)		Note4
DB25.13	POWER		Controllable 3.3V, 180mA power
DB25.25	GND		

- Note:
1. PP: Push-Pull, capable of drive -3mA/8.5mA at 2.6V/0.6V.
 2. ODI: Open-Drain high output with weak pull-up(66KOhm min.). It acts as an input.
 3. All port I/O pins are 5V tolerant,
 4. These pins are defined as ground in a standard PC printer interface device. Care has be taken to avoid a short condition when the USB2Any was plugged to a such device in the SIO mode. In SPP/EPP mode, it is safe because they are set at OD state.
 5. In SPP/EPP mode, this signal on the DB25 header is inverted from the data writ/read to the PC printer port register. However it is not the case for the USB2Any, where they have the same polarity.
 6. These two pins has to be pull-up externally by resistor 2.7K~5.6K to make the I2C works.
 7. In SIO mode, the SPI is set to 4-wire single-master and CKPHA=CKPOL=0, the I2C is set to master.
 8. The internal micro controller chip is a Silicon Laboratory's C8051F320. Please refer its data sheet for more detailed information.

3.0 USB2Any User Interface:

The USB2Any device is of the USB 2.0 HID class that uses one IN report and one OUT report. Both reports are 64 bytes in size.

Device commands are embedded in the COMMAND_REPORT and sent to the USB2Any. It is an OUT report, start by two lead codes 0x5A and 0xA5, followed by the commands with or without parameters, and ended with an end code 00. The valid command number(Cmd#) starts from value 0x01 and up.

COMMAND_REPORT(OUT) format:

Lead Codes		Command 1			Command 2		Command 3			End Code	Not used area
0x5A	0xA5	Cmd#	para1	Cmd#	Cmd#	00	
----- 64 bytes -----											

The following example set the port bit P2.1 and read back the port P1. Its command report will look like:

0x5A	0xA5	0x21	0xA1	0x03	0x90	00
Lead Code	CmdPBitSet	bit P2.1	CmdSFRRead	P1	End Code		

Each COMMAND_REPORT always has a STATUS_REPORT returned. It is an IN report with one lead code 0xA5, followed by one status byte and then the read back data(if there is any). User should check the status byte to see if this COMMAND_REPORT was executed correctly.

STATUS_REPORT(IN) format:

Lead Code	Status Byte	
0xA5	status	Data, data ... ,
----- 64 bytes -----		

The status byte has the following meanings:

bit7=1: an error has occur during the execution of COMMAND_REPORT.

“bit[5:0] minus 1” is the offset location where the error occurs in the COMMAND_REPORT.

Some possible errors are:

1. Not a valid COMMAND_REPORT, the lead codes(0x5A, 0xA5) was missed.
2. Not a valid command code(Cmd#) in the COMMAND_REPORT.
3. CmdWaitWIPL command timeout error on waiting the WIP bit of a 25 series SPI flash devices.
4. ACK signal fails on an I2C slave device.
5. 10ms timeout error.

bit7=0: Successful

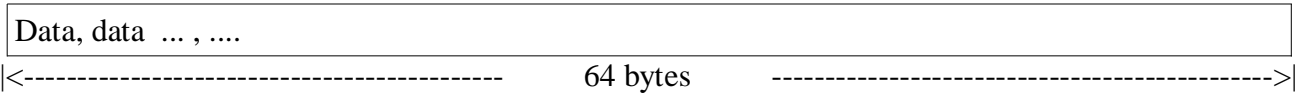
Data returned in the STATUS_REPORT starts from the third byte. If there are multiple commands in one COMMAND_REPORT has their data returned, these data are just piled up byte-by-byte into the STATUS_REPORT. User has to separate them by his knowledge about which command gets which bytes back.

For the above example, its status report should look like this(suppose the P1's data is 0x65):

0xA5	0x00	0x65
Lead Code	status	P1 data	

There are another kind of usage of the IN/OUT report which carries pure data, the DATA_REPORT.

DATA_REPORT format:



Large amount of data can be sent from PC to device through multiple "OUT" DATA_REPORT by executing the "CmdLoadReportN" command. The user has to send exact amount of DATA_REPORT (maximum number is 12) this command expects. All data transferred(64*12=768 bytes maximum) was saved in the USB2Any's page buffer(XDATA, start from address 0x00C0) for later use.

The following example send one IC page(128 bytes, two reports) of data to the device, the COMMAND_REPORT will be look like:

0x5A	0xA5	0x0B	02	00
Lead Code	CmdLoadReportN #Report		End Code		

after this COMMAND_REPORT was sent out, two DATA_REPORT must follows:



Then the STATUS_REPORT can be read back.

The device's data can also be read massively and send back to the PC through multiple IN DATA_REPORT by executing "CmdSPI0ReadReportL" or "CmdSMB0ReadReportL" command. The user should extract the exact amount of report before getting the final STATUS_REPORT.

The following example read 0x0100 reports back from a SPI device, the COMMAND_REPORT is like:

0x5A	0xA5	0x15	0x01	0x00	00
Lead Code	CmdSPI0ReadReportL		HI byte	LO byte	End Code	

after the COMMAND_REPORT was sent out, the 0x0100 data reports has to be read:



...



then the final STATUS_REPORT can be read.

When any command in one COMMNAD_REPORT fails, the whole report will be aborted immediately and the returned STATUS_REPORT will show its error location. The followings are those commands which can possibly induce an error and was indicated by an [Err] label in its command descriptor:

- CmdSMB0WriteN and CmdSMB0UnloadN : error when ACK bit check fails
- CmdWaitWIPL : error when the WIP bit of the status register in a SPI flash device doesn't return to its normal status within the timeout period.
- CmdADC0Get, SPP/EPP strobed read/write and "wait bit" commands : error on 10ms timeout.
- CmdSFRWrite, CmdPBitSet, CmdPBitClr: access to an invalid address.

4.0 Host Command List(V1.01):

int OpenU2A(void);

Find an USB2Any device on the PC. If there is one and was opened successfully, it will return a zero. Also the maximum wait time for read/write a report was set to 5 seconds.

int CloseU2A(void);

Close the opened USB2Any device. If closed successfully, it will return a zero.

*int ReadReport(char *ptr);*

Read an IN report and stored at the buffer pointed to by the ptr pointer. The report can be a DATA_REPORT or a STATUS_REPORT, depends on commands sent in the COMMAND_REPORT. The buffer must be 64 bytes long. If read successfully, it will return a zero.

*int WriteReport(char *ptr);*

Write an OUT report with data stored at the buffer pointed to by the ptr pointer. This report can be a DATA_REPORT or a COMMAND_REPORT. The buffer must be 64 bytes long. If write successfully, it will return a zero.

void SetMaxWaitTime(unsigned int ms);

Set the maximum wait time(in mS unit) for report read/write to complete.

5. Device Command List(V1.01):

Each command was listed below in “**CmdName(Value) [+ para + para + ..], Bytes, [Err]**” format where:

CmdName: the name which should be used in the programming, rather than using its value directly.

Value: a number which represents this command.

[para]: the parameters this command requires. It is optional.

Bytes: how many bytes this command will take totally, including its parameters.

[Err]: this command may cause error during its execution. If it happens, will abort the whole command report execution.

5.0.1 General purpose commands:

CmdGetVer(0x01), 1 byte

Get the USB2Any two bytes firmware version and returned in the STATUS_REPORT. A value of 0x0100 means version 01.00.

Command_report:	0x01
-----------------	------	------	------

CmdPowerOn(0x02), 1 byte

Turn on 3.3V/180mA voltage regulator to supply power to the external circuit.

Command_report:	0x02
-----------------	------	------	------

CmdPowerOff(0x03), 1 byte

Turn off 3.3V/180mA voltage regulator. This pin has a 1K//2.2uF pull-down.

Command_report:	0x03	...
-----------------	------	------	-----

CmdSFRRead(0x04) + reg, 2 bytes

Read SFR register and store it in the STATUS_REPORT. The "reg" should be between 0x80~0xFF. For a non-existent register, a 0xFF will always be returned.

Command_report:	0x04	reg	
-----------------	------	------	-----	--

CmdSFRWrite(0x05) + data + reg, 3 bytes, [Err]

Write data to SFR register. The "reg" should be between 0x80~0xFF and exists. For value below 0x80, a non-existent SFR or a forbidden SFR(see below), an error will occur.

Some of the USB2Any related SFR registers shouldn't be altered or the system may be crashed. These registers are listed here for reference:

SYSTEM registers: SP, RSTSRC.

INTERRUPT registers: IE, IP, EIE1, EIP1, EIE2, EIP2, ITO1CF.

OSCILLATORS registers: OSCXCN, OSCICN, OSCICL, CLKSEL, CLKMUX.

USB registers: USB0ADR, USB0DAT.

Command_report:	0x05	data	reg	
-----------------	------	------	------	-----	--

CmdSetPWidth(0x06) + N, 2 bytes

Set the pulse width to $(41.6\text{ns} * (17 + (N - 1) * 4))$ where N is 0~255(0 means 256). The minimum width is ~0.71uS(default). The TCK(P2.0), nSTB(P2.0), nDSTB(P2.1) and nASTB(P2.3) pulses will be influenced.

Command_report:	0x06	N
-----------------	------	------	---	------

CmdWaitN100uS(0x07) + N, 2 bytes

Wait N number of 100uS. N is 0 to 255(0 means 256).

Command_report:	0x07	N
-----------------	------	------	---	------

CmdSelEPP(0x08), 1 byte

Set USB2Any to EPP mode(also the power-on default). Both port0 and port1 are open-drain pull-up. No SPI, I2C or UART functions are available. Port2 bit0~3 is push-pull output low, bit4~7 is open-drain pull-up.

Command_report:	0x08
-----------------	------	------	------

CmdSelSPP(0x09), 1 byte

Set USB2Any to SPP mode. Port 1 is push-pull output low. Port 0 is open-drain pull-up. No SPI, I2C or UART functions are available. Port2 bit0~3 is push-pull output low, bit4~7 is open-drain pull-up.

Command_report:	0x09
-----------------	------	------	------

CmdSelSIO(0x0A), 1 byte

Set USB2Any to SIO mode. Port 1 is open-drain pull-up. Port 0 works for SPI, I2C and UART functions. Port2 bit0~3 is push-pull output low, bit4~7 is open-drain pull-up.

CmdLoadReportN(0x0B) + #Report, 2 bytes

Store massive data into USB2Any's page buffer(in XDATA, start at 0x00C0, 768 bytes max.) through multiple OUT_DATA_REPORT. The number of reports is 1 to 12. After this command was completed, the page buffer pointer will be reset to point to the beginning of this buffer for later usage by commands "CmdSPI0UnloadN" and "CmdSMB0UnloadN".

Command_report:	0x0B	#Report
-----------------	------	------	---------	------

5.0.2 SPI commands:

CmdSPI0SetRate(0x0C) + value, 2 bytes

Set the SPI clock by the following formula. The "value" will be loaded into the 8051's SPI0CKR register. $F_{sck}=24\text{MHz}/(2*(\text{SPI0CKR}+1))$. The default SCK is 2MHz with SPI0CKR=5.

Command_report:	0x0C	value
-----------------	------	------	-------	------

CmdSPI0En(0x0D), 1 byte

Set SPI mode to 4-wire single-Master, CKPHA=CKPOL=0, $F_{sck}=2\text{MHz}(@24\text{MHz SYSCLK})$ and enable it. The related 8051's registers were set to SPI0CKR=5, SPI0CFG=0x40, SPI0CN=0x0F

Command_report:	0x0D
-----------------	------	------	------

CmdSPI0Dis(0x0E), 1 byte

Disable SPI. This command only clears the SPIEN bit of the SPI0CN.

Command_report:	0x0E
-----------------	------	------	------

CmdSPI0CS0_Byte(0x0F) + data, 2 bytes

Set NSS to 0 to enable the SPI device and then send out the data byte.

Command_report:	0x0F	data
-----------------	------	------	------	------

CmdSPI0CS1(0x10), 1 byte

Wait for SPIF=1 and then set NSS to 1 to disable the SPI device. SPIF is bit 7 of the control register SPI0CN.

CmdSPI0WriteN(0x11) + N + data1 + data2 + ... dataN, N+2 bytes

Write N bytes of data to SPI slave. The length N should be limited not to exceed the report buffer.

Command_report:	0x11	N	data1	Data2	dataN
-----------------	------	------	---	-------	-------	------	-------	------

CmdSPI0ReadN(0x12) + N, 2 bytes

Read N bytes of data from SPI slave. The read data were stored in the STATUS_REPORT. The length N should be limited not to exceed the report buffer.

Command_report:	0x12	N
-----------------	------	------	---	------

CmdWaitWIPL(0x13) + Len_Hi + Len_Lo, 3 bytes, [Err]

Wait for WIP bit to be cleared in a timeout period of (Len * 10) ms. For example, if Len_Hi=0x01 and Len_Lo=0x20, the timeout period will be 288(0x0120) x 10ms = 2880ms. COMMAND_REPORT was aborted if timeout occurs. This command is only meaningful to some serial flash devices.

Command_report:	0x13	Len_Hi	Len_Lo
-----------------	------	------	--------	--------	------

CmdSPI0UnloadN(0x14) + N, 2 bytes

Write N bytes(usually one page) of data stored in the USB2Any's page buffer to SPI slave, start from the address pointed to by the page buffer pointer. The pointer will be advanced to point to the next available data after this command completed. N is 0 to 255(0 means 256).

Command_report:	0x14	N
-----------------	------	------	---	------

CmdSPI0ReadReportL(0x15) + Len_Hi + Len_Lo, 3 bytes

Read "Len" number of reports of data from SPI slave and send back to PC through IN DATA_REPORT. The "Len" is a 16bit value. If Len=0x0100, then 256 reports(64 x 256 = 16,384 bytes) will be sent back.

Command_report:	0x15	Len_Hi	Len_Lo
-----------------	------	------	--------	--------	------

5.0.3 I2C commands:

Note: Important! The SCL and SDA pin should be pull-up externally by a 2.7K ~ 5.6K resistor.

CmdSMB0SetRate(0x16) + value, 2 bytes

Set the bit rate(at 66/33 duty cycle) of I2C by equation BitRate=Ft2_overflow/3. The "value" was loaded into timer2's TMR2RLH to decide its overflow distance. The default is 100KHz (value=0xB0, 80 clocks to overflow) with timer2 running at 24MHz. Change the value to 0x60 will slow down clock to 50KHz.

Command_report:	0x16	value
-----------------	------	------	-------	------

CmdSMB0En(0x17), 1 byte

Set SMB0CF=0xD1 to initialize the SMB0 function and run Timer2.

Command_report:	0x17
-----------------	------	------	------

CmdSMB0Dis(0x18), 1 byte

Set SMB0CF=0x51, SMB0CN=0 to stop the SMB0 function and Timer2.

Command_report:	0x18
-----------------	------	------	------

CmdSMB0Start(0x19), 1 byte

Set STA=1 to trigger a "START" condition.

Command_report:	0x19
-----------------	------	------	------

CmdSMB0Stop(0x1A), 1 byte

Set STO=1 and SI=0 to trigger a "STOP" condition. Usually used to end an I2C write session.

Command_report:	0x1A
-----------------	------	------	------

CmdSMB0NACKStop(0x1B), 1 byte

Clear ACK(a NACK condition) and jump to "CmdSMB0Stop". Usually used to end an I2C read session.

Command_report:	0x1B
-----------------	------	------	------

CmdSMB0ReadN(0x1C) + N, 2 bytes

Read N bytes of data from I2C slave and stored in the STATUS_REPORT. Set ACK to 1(an ACK condition) after each byte was read. The length N should be limited not to exceed the report buffer.

Command_report:	0x1C	N
-----------------	------	------	---	------

CmdSMB0Read_NACK(0x1D), 1 byte

Read the last byte of data and ended with a NACK condition(ACK=0). Usually a "CmdSMB0Stop" command will follow to end the read session.

This command is not really needed anymore. Instead, using the "CmdSMB0NAckStop" command.

Command_report:	0x1D
-----------------	------	------	------

CmdSMB0WriteN(0x1E) + N + data1 + data2 ...+ dataN, N+2 bytes, [Err]

Write N bytes of data to I2C slave. Check ACK after each data written and abort immediately if not an ACK. The length N should be limited not to exceed the report buffer. It's usually for device page size <= 32 bytes.

Command_report:	0x1E	N	data1	Data2	dataN
-----------------	------	------	---	-------	-------	------	-------	------

CmdSMB0UnloadN(0x1F) + N, 2 bytes, [Err]

Write N bytes of data(usually one page) stored in the USB2Any's page buffer to I2C slave, start from the address pointed to by the page buffer pointer. The pointer will be advanced to next available data after this command completed. Check ACK after each data written and abort immediately if it is not an ACK. N is 0 to 255 (0 means 256).

Command_report:	0x1F	N
-----------------	------	------	---	------

CmdSMB0ReadReportL(0x20) + Len_Hi + Len_Lo, 3 bytes

Read "Len" number of reports of data from I2C slave and send back to PC through IN DATA_REPORT. The "Len" is a 16bit value.

Command_report:	0x20	Len_Hi	Len_Lo
-----------------	------	------	--------	--------	------

5.0.4 SPP/EPP Port I/O commands:

Note1. Pulse are generated by toggling bit. It should be initialized to a proper level to get a correct direction.

Note2. Port access can be completed through SFR read/write command. Their address was listed below:

Port's data: P0=0x80, P1=0x90, P2=0xA0.

Port's output configuration: P0MDOUT=0xA4, P1MDOUT=0xA5, P2MDOUT=0xA6.

1 for push-pull, 0 for open-drain.

CmdPBitSet(0x21) + bit, 2 bytes, [Err]

Set bit of port P0, P1 and P2. The valid bit address should be within 0x80~0x87, 0x90~0x97, 0xA0~0xA7.

Invalid address will cause an error.

CmdPBitClr(0x22) + bit, 2 bytes, [Err]

Clear bit of port P0, P1 and P2. The valid bit address should be within 0x80~0x87, 0x90~0x97, 0xA0~0xA7.

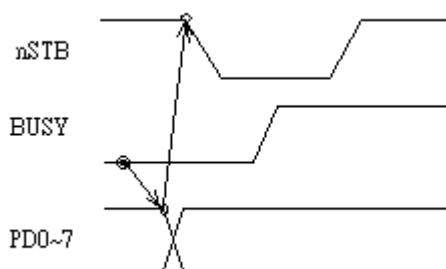
Invalid address will cause an error.

CmdStbWrite(0x23) + data, 2 bytes, [Err] // for SPP mode

Wait for BUSY=0(P0.7) with 10ms timeout, write "data" to Port1, toggle nSTB(P2.0) for a pulse which its width can be set through command "CmdSetPWidth". The default pulse width is ~0.71uS.

Command_report:

....	0x23	data
------	------	------	------



CmdStbWriteN(0x24) + N + data1 + data2 ...+ dataN, N+2 bytes, [Err] // for SPP mode

Write N bytes of data through repeat calling the command "CmdStbWrite". The length N should be limited not to exceed the report buffer.

Command_report:

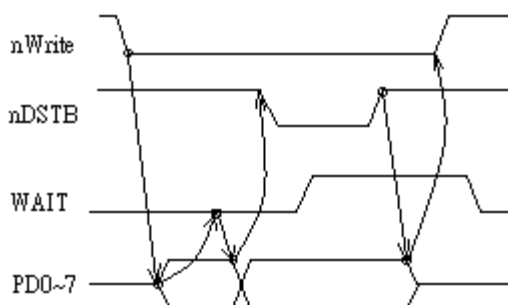
....	0x24	N	data1	Data2	dataN
------	------	---	-------	-------	------	-------	------

CmdDStbWrite(0x25) + data, 2 bytes, [Err] // for EPP mode Data Write

Set nWrite=0(P2.0), Change Port 1 to push-pull mode, wait for WAIT=0(P0.7) with 10ms timeout, write "data" to Port 1, toggle nDSTB(P2.1) for a pulse which its width can be set by command "CmdSetPWidth", Change Port 1 back to open-drain pull-up mode, and then set nWrite=1. The default pulse width is ~0.71uS.

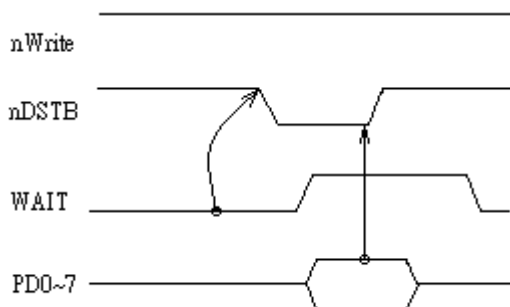
Command_report:

....	0x25	data
------	------	------	------



CmdDStbRead(0x26), 1 byte, [Err] // for EPP mode Data Read

Wait for WAIT=0(P0.7) with 10ms timeout, toggle nDSTB(P2.1) for a pulse which its width can be set by command "CmdSetPWidth", read data from Port 1 just before the pulse ends, and store data into the STATUS_REPORT. If timeout error occurs, a dummy data will be stored in the STATUS_REPORT. The default pulse width is ~0.71uS.



CmdAStbWrite(0x27) + data, 2 bytes, [Err] // for EPP mode Address Write

It is same as the above command "CmdDStbWrite" except that the strobed pin is nASTB(P2.3).

CmdAStbRead(0x28), 1 byte, [Err] // for EPP mode Address Read

It is same as the above command "CmdDStbRead" except that the strobed pin is nASTB(P2.3).

CmdWaitBit0(0x29) + MASK, 2 bytes, [Err]

Wait for all the bits in {P0.7, P0.6, P0.5, P0.4, P0.3, P2.6, P2.5, P2.4}, which its relative bit position in the MASK byte was set, to become zero. This wait has a 10ms timeout and will cause error if it occurs.

For example, waiting for both bit P0.6 and P2.4 to become zero, the MASK should be set to 0b01000001.

Command_report:

....	0x29	MASK
------	------	------	------

CmdWaitBit1(0x2A) + MASK, 2 bytes, [Err]

Same as the above command except wait for bits to become one.

5.0.5 JTAG commands:

Note: The TCK pulse was generated through bit toggle so a correct initial value should be set before use. Its default pulse width is ~0.71uS and can be changed by command "CmdSetPWidth".

CmdJTAGReset(0x2B), 1 byte

Go to "Run-Test/Idle" state.

CmdTCK_TMS1(0x2C), 1 byte

Set TMS=1 and send out one TCK pulse. TDI is intact and TDO is ignored.

CmdTCK_TMS0(0x2D), 1 byte

Set TMS=0 and send out one TCK pulse.

CmdTDI1(0x2E), 1 byte

Set TDI=1.

CmdTDI0(0x2F), 1 byte

Set TDI=0.

CmdTCKsL(0x30) + Len_Hi + Len_Lo, 3 bytes

Send "Len" number of TCK pulses. The TDI and TMS is intact and the TDO is ignored.

CmdBitsRWExitL(0x31) + Len_Hi + Len_Lo + data1 + data2 + ... + dataN, N+3 bytes

Write(to TDI) "Len" number of bits which were embedded in the COMMAND REPORT and, at the same time, read(from TDO) the same length of data back and stored in the STATUS REPORT. The first bit is bit 0 of the first byte, the second bit is bit 1, and so on. Each bit was read/write before TCK pulse starts and the last TCK pulse will go with TMS=1 so that it will exit from the current state. Usually this command was invoked when JTAG is in the "Shift-DR" or "Shift-IR" state, and it will end in the "Exit1-DR" or "Exit1-IR" state.

Command_report:

....	0x31	Len_Hi	Len_Lo	Data1	Data2	dataN
------	------	--------	--------	-------	-------	------	-------	------

Note: How many bits a command report can carry depends on how many commands it has. The largest space left for data is when this command is the only one in a command report. So "the report length 64 minus 3 bytes of overhead(2 lead codes, 1 end code), minus 3 bytes of command itself" is, 58 bytes or 464 bits.

CmdBitsRWL(0x32) + Len_Hi + Len_Lo + data1 + data2 + ... + dataN, N+3 bytes

Same as the above command except that the last TCK pulse will keep the TMS intact to remain in its current state.

5.0.6 ADC commands:

CmdADC0En(0x33), 1 byte

Set the P2.5 and P2.6 to analog input. The reference voltage is from VDD, the conversion's code format is left-justified, the SAR clock is 2MHz.

CmdADC0Dis(0x34), 1 byte

Reset the P2.5 and P2.6 to open-drain pull-up input. All ADC related registers reset to its power-on status.

CmdADC0SetAMX(0x35) + AMX0P + AMX0N, 3 bytes

Select the analog input channel.

Set AMX0P=0x0E to select P2.6 as the input channel, or AMX0P=0x0D to select P2.5. Always set the AMX0N=0x1F for single-ended mode.

CmdADC0Get(0x36), 1 byte, [Err]

Start a software trigger conversion. The result ADC value(10 bits) will be stored in the STATUS_REPORT in two adjacent bytes. Bit9~2 is the first byte and bit1~0 was left-justified in the second byte.

5.0.7 RS232(TTL) commands:

Cmd232En(0x37), 1 byte

Enable RS232 with default baud rate 115200. The SIO mode has to be selected to make it works.

Cmd232Read_Flag(0x38), 1 byte

Check if there is a data available and return it in the STATUS_REPORT. If there is a data, the flag 0xFF was stored first and then the data byte follows. If not, only the flag 0x00 was stored.

Command_report:

....	0x38
------	------	------

Status_report(has data):	0xFF	data
Status_report(no data):	0x00	

Cmd232ReadN(0x39) + N, 2 bytes

Read N bytes of data and stored in the STATUS_REPORT. The length N should be limited not to exceed the report buffer.

Cmd232WriteN(0x3A) + N + data1 + data2 + ... + dataN, N+2 bytes

Write N bytes of data. The length N should be limited not to exceed the report buffer.

Cmd232SetBaud(0x3B) + data0 + data1, 3 bytes

Load data0 into CKCON and data1 into TH1 to set the RS232 baud rate. The default baud rate is 115200.

Below are values(data0, data1) for different baud rate:

0x22, 0x98 for 2400 Baud, 0x20, 0x30 for 4800, 0x20, 0x98 for 9600, 0x21, 0x64 for 19200, 0x21, 0xB2 for 38400 and 0x28, 0x98 for 115200.

5.0.8 Power user commands:

CmdUserReadPage(0x3C), 1 byte

Read the user's 512 bytes of 8051's \$2E00~\$2FFF code space through 8 IN DATA_REPORT and sent to PC. This command is only for code verification purpose.

CmdUserWritePage(0x3D), 1 byte

The user code page(\$2E00~\$2FFF) was erased first and then 512 bytes of user code was written. These codes must be packed into 8 OUT DATA_REPORT and sent to USB2Any from PC during this command execution. The codes can be build by any general assembler tool, but have to follow some rules which were stated in the template file "User.asm".

User defined commands.

CmdUser0(0x3E), CmdUser1(0x3F), CmdUser2(0x40), CmdUser3(0x41), CmdUser4(0x42), CmdUser5(0x43), CmdUser6(0x44), CmdUser7(0x45), CmdUser8(0x46), CmdUser9(0x47)

6.0 Examples:

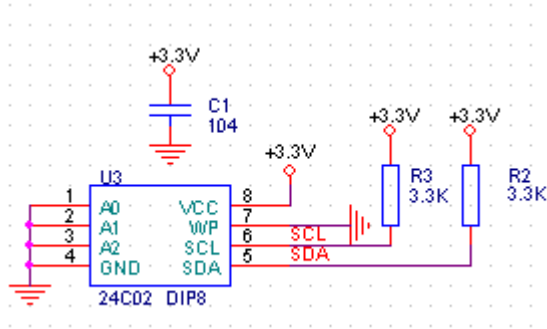
The main file is USB2Any.cpp. For each test, the last “#include” statement should be modified to the right filename. For example, on testing I2C, this statement should be “#include ConsI2C.cpp”.

6.0.1 I2C

Description:

Download/Upload AT24C04 or AT24C512 EEPROM from/to a binary file. This example shows two ways on handling between different page size device, one is for size smaller than 64 bytes which a whole page can be embedded into one command report, the other is for size larger than or equal to 64 bytes which the page data was sent separately through multiple data reports.

Circuit:



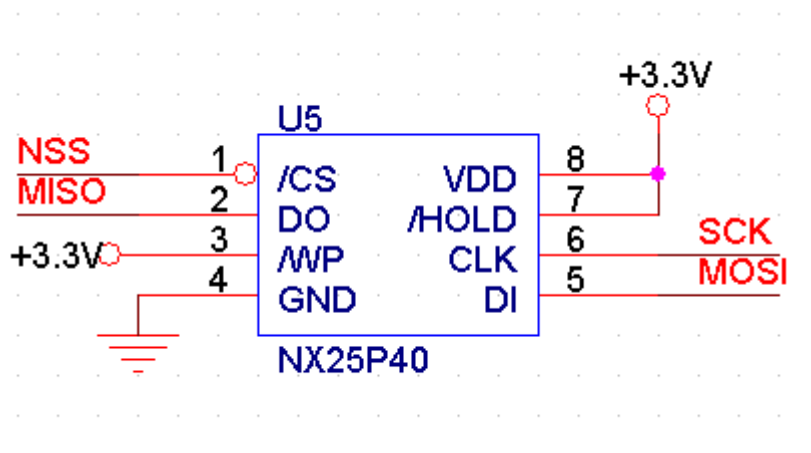
Required files: U2ACmds.h, ConsComm.cpp, ConsI2C.cpp, USB2Any.cpp

6.0.2 SPI:

Description:

Download/Upload MX25L1005 or MX25L4005 serial flash from/to a binary file.

Circuit:



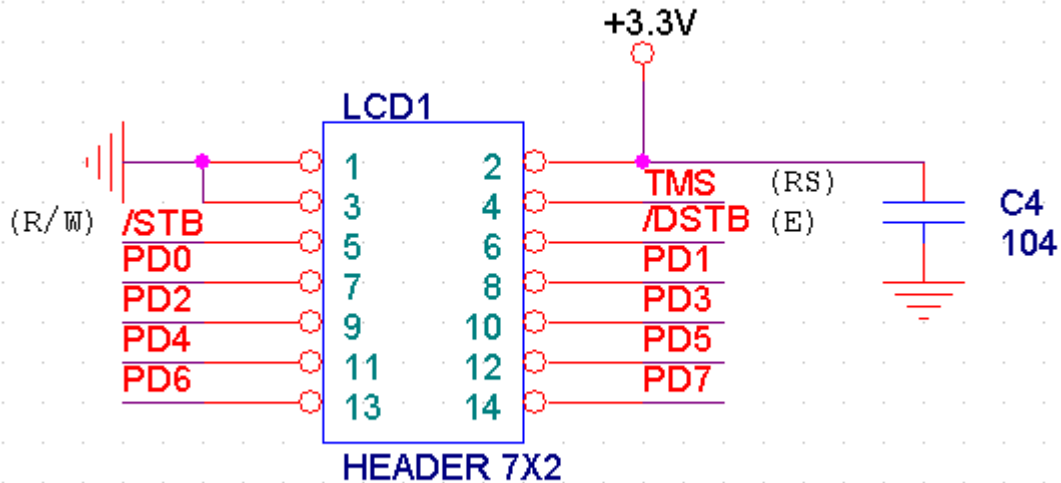
Required files: U2ACmds.h, ConsComm.cpp, ConsSPI.cpp, USB2Any.cpp

6.0.3 EPP:

Description:

Using commands “CmdDStbWrite” and “CmdDStbRead” to write/read a 16x2 character LCD module.

Circuits:



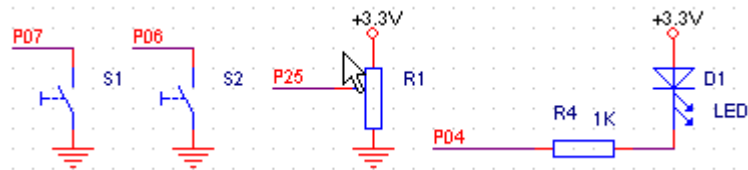
Required files: U2ACmds.h, ConsComm.cpp, ConsEPP.cpp, USB2Any.cpp

6.0.4 GPIO/ADC:

Description:

A simple digit I/O example to handle switches and LED lamp, and a single-ended analog input.

Circuits:



Required files: U2ACmds.h, ConsComm.cpp, ConsIO.cpp, USB2Any.cpp

6.0.5 JTAG:

Description:

Find out how many devices are in the JTAG chain and read their ID. The device used here for testing is a XCR3128XL-VQ100 CPLD.

Circuits:

The wiring is straightforward.

Required files: U2ACmds.h, ConsComm.cpp, ConsJTAG.cpp, USB2Any.cpp