

All aspects of design including pinout, dimensions and software syntax are Copyright 2010-2011 Itron UK Limited A subsidiary of Noritake Co. Ltd Japan

**Product No** TU640x480C-XXX  
**Issue Date** 1/4/2011  
**Document Ref** 42781

Index	Description	Section
	<b>General</b>	1
	Dimensions	
	Optical and Environmental Parameters	
	Electrical Parameters	
	Connector Pin Assignment	
	Jumper and additional Connector information	
	PCB (rear view)	
	<b>Accessories</b>	2
	USB Cable, RS232 Cable, CAN Bus Interface, Battery Holder, IDC Interface Cable, AC97 Audio Module, USB-SD expander	
	<b>Overview</b>	3
	<b>System Hardware Setup Parameters and Development Status</b>	4
	System, RTC and Counter Setup	5
	RS232 Interface	6
	RS485 Interface	7
	CMOS Asynchronous Interfaces	8
	SPI Interface	9
	I2C Interfaces	10
	Keyboard and I/O Interfacing	11
	<b>Command Overview</b>	12
	<b>System Commands</b>	13
	FPROG.....FEND LIB(Name,Source) INC(Source) Reset(Name) ; ;; [cmd(...);cmd(...);...cmd(...);]	
	<b>Page and Group Commands</b>	14
	PAGE(Name,Style){....} LOAD(Dest,Name,Name,...) SHOW(Name) HIDE(Name) DEL(Name)	
	<b>Commands for Cursor Position, Text, Draw, Image and Keys</b>	15
	POSN(X,Y,Page/Name,Style) TEXT(Name,Text Style) DRAW(Name,X,Y,Style) IMG(Name,Source,X,Y,Style) KEY(Name,Function,X,Y,Style)	
	<b>Function Commands</b>	16
	VAR(Name,Style) IF(Var~Var?Function1:Function2) LOOP(Name,Var1){.....} INT(Name,Buffer,Function) CALC(Result,VarA,VarB,Method) FUNC(Name){....} RUN(Name) WAIT(Time)	
	<b>Reserved Word</b>	17
	<b>Styles List</b>	18
	<b>Setup List</b>	19
	<b>Character Fonts</b>	20
	<b>Colour Chart</b>	21
	<b>Getting Started</b>	22
	<b>Example Project - Elevator</b>	23
	<b>FAQ</b>	24

# iSMART Noritake Itron 5.7" TFT Module




## General - 1

### 5.7" iSMART TFT Module

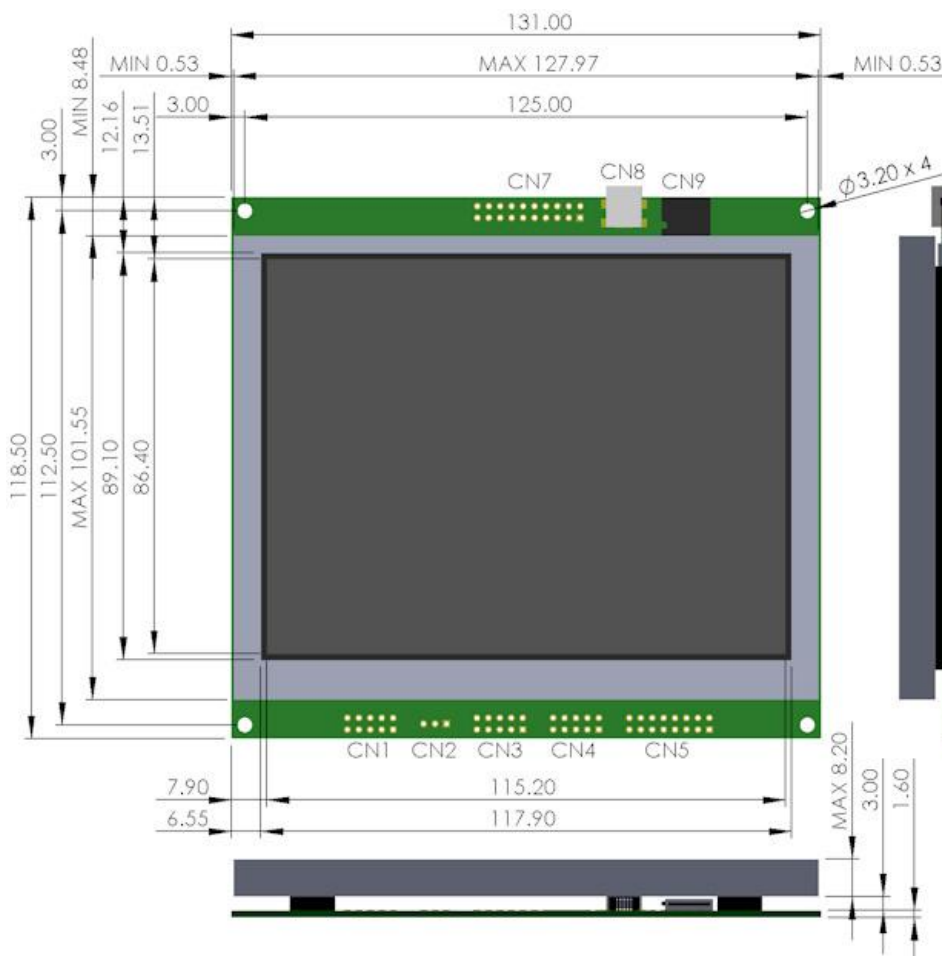
640X480 pixels  
 262,144 Colours (18 bit)  
 60 Page Display RAM  
 128M Byte Flash  
 4G+ Micro SDHC Slot  
 LED Backlight Control  
 5V Supply 3.3V Logic  
 ASCII + UNICODE Fonts

Full RS232 Port  
 SPI - I2C Interfaces  
 Sync Serial Controller  
 USB 2.0 Interface  
 Resistive Touch Screen  
 Up to 12 x 12 Key Control  
 Up to 24 User Digital I/O  
 2 Analogue Inputs  
 2 PWM Outputs  
 Real Time Clock + Date  
 Run Animations  
 Auto Menu Control  
 Screen Rotation - 90, 180  
 Graphic User Interface  
 Object Oriented Program  
 Integrated Debugger

#### Downloads

Full Specification (pdf)   
 Full Specification (compiled)   
 2D Mechanical 

The 8.2mm TFT thickness includes a touch screen. This dimension is reduced for non touch versions



This product has been designed to simplify the implementation of TFT technology into your product. The high level text based object oriented command structure, entity library and 100 page screen memory allow most of the processing to be undertaken by the TFT module leaving the host CPU to concentrate on the core application processes. This allows proven firmware running on small 8 bit microcontrollers to be modified to drive this TFT module with a minimum of work and risk.

Module Part Number	Price**	RS232	RS485	Touch	USB CN8	Battery Holder	CANBUS Adaptor	Note
TU640X480C-K612A1		Yes	-	-	-	-	-	-
-K612A1T	€134	Yes	-	Yes	-	-	-	Standard
-K612A1TU*	€135	Yes	-	Yes	Yes	-	-	Can use as Dev Kit
-K612A1TUB	€136	Yes	-	Yes	Yes	Yes	-	Battery not included
-K612A1TUBC	€150	Yes	-	Yes	Yes	Yes	Yes	Battery not included
TU640x480C-K611A1		Yes	Yes	-	-	-	-	-
-K611A1T	€140	Yes	Yes	Yes	-	-	-	-
-K611A1TU	€141	Yes	Yes	Yes	Yes	-	-	-
-K611A1TUB	€142	Yes	Yes	Yes	Yes	Yes	-	Battery not included
-K611A1TUBC	€156	Yes	Yes	Yes	Yes	Yes	Yes	Battery not included

\* Main distributor stock item, other versions supplied to order. Pre-fitted connector options are available.

\*\* Unit price excludes freight and VAT. Option to pay in GBP £ , Orders will be delivered from December 20th

#### Optical & Environmental Parameters

Screen Type	640x480 pixels - RGB Stripe - Pixel Pitch 0.18x0.18mm
Display Area	115x86mm - 5.7" diagonal
RGB Colours	262,144 (18 bit)
Display Type	Transmissive
Contrast Ratio	250:1
View Angle (typ)	60 degrees
Response Time	25ms @ 25C
Default Viewing Angle	12 o'clock (6 o'clock-Invert the PCB and set 180 degrees orientation in software)
Operating Temperature	-20C to +70C
Storage Temperature	-30C to +80C
Humidity	20% to 70% RH
Vibration	10-55-10Hz, all amplitude 1mm, 30Min., X-Y-Z (Non operating)
Shock	392m/s <sup>2</sup> (40G) 9mS X-Y-Z, 3 times each direction (Non operating)

# iSMART Noritake Itron 5.7" TFT Module

## Electrical Parameters

Parameter	Sym	Min	Typ	Max	Unit	Condition	Note	
Supply Voltage	VCC	4.5	5	5.5	VDC	VSS=0V	Absolute Max 6.0VDC	
Logic Supply Output	VDD	3.2	3.3	3.4	VDC	VCC=5V	Max50mA	
Logic Input Voltage	"H"	VIH	-0.5	-	3.4 (1)	VDC	VCC=5V	/RES, K0-K24, SCK, /SS, HB, SIN, SCL,SDA
	"L"	VIL	VSS	-	VSS+0.5	VDC	VSS=0V	
Logic Output Voltage	"H"	VOH	3.0	-	3.4	VDC	IOH=2mA VCC=5v	K0-K24, SDA, SCL, SOUT, MB
	"L"	VOL	0	-	0.7	VDC	IOL=-2mA VCC=5V	
"H" Level Logic Input Current	IIH	-	-	1.0	uADC	VCC=5.5V	/RES, K0-K24, SCK, /SS, SIN, SCL, SDA	
"L" Level Logic Input Current	IIL	-	-	1.0	uADC	VCC=5.5V		
RS232 Input Voltage	"H"	VIH	2	-	15	VDC	VCC=5V	RXD, CTS, DSR
	"L"	VIL	-15	-	VSS+0.5	VDC	VCC=5V	
RS232 Output Voltage	"H"	VOH	4	7	-	VDC	3kΩ to GND VCC=5V	TXD, DTR, RTS
	"L"	VOL	-	-3	-2	VDC	3kΩ to GND VCC=5V	
Power Supply Current 1	ICC1	450	480	550	mADC	VCC=5V	All dots on	
Power Supply Current 2	ICC2	120	140	170	mADC	VCC=5V	LED Backlight Off	
Power Supply Current 3	ICC3	50	60	70	mADC	VCC=5V	Reset LOW	

Note (1) The voltage applied to logic signals must not exceed the rising VCC at power on as this could affect module initialisation

## Connector Pin Assignment

CON	Function	1	2	3	4	5	6	7	8	9	10	Note
CN1	RS232 Port	NC	DTR	TXD	CTS	RXD	RTS	DSR	NC	GND	5V	Fits 9 way IDC D type pin 1-9
	RS232+RS485	T+	R-	TXD	CTS	RXD	RTS	R+	T-	GND	5V	Available on -K611xxx
CN2	5V Power In / Piezo to GND	5V	/PZ	0V	-	-	-	-	-	-	-	Connect piezo negative
	I2C Serial Mode	5V	SCL	-	SDA	0V	-	-	/RES	MB	HB	3v3 Logic (5v in option)
CN3	Asynchronous Serial Mode	5V	-	SI	-	0V	-	SO	/RES	MB	HB	3v3 Logic (5v in option)
	Clock Serial / SPI Mode	5V	SCK	/SS	MOSI	0V	MISO	/IRQ	/RES	MB	HB	/IRQ flags read request to host
CN4	Analogue In, PWN Audio	AN1	AN2	0V	5V	PW1	PW2	ATX	ARX	ACK	AFS	AC97 Audio Pins 7-10
	User I/O	K16	K17	0V	5V	K18	K19	K20	K21	K22	K23	Additional I/O

CON	Function	1/2	3/4	5/6	7/8	9/10	11/12	13/14	15/16	17/18	19/20	Note
CN5	USB/ SD Card Extension	DA2	CDA	CK	DA0	0V	0V	DM	CNX	-	-	SD Card Pins 1-10 USB Pins 11-16
		DA3	3V3	0V	DA1	CD	5V	DP	0V	-	-	
CN7	8x8 Keyboard Matrix and user I/O Ports	5V	3V3	K0	K2	K4	K6	K8	K10	K12	K14	3V3 output max 50mA
		0V	0V	K1	K3	K5	K7	K9	K11	K13	K15	

CN8	USB Connector	Standard Mini B can be omitted on user request. 5V power is then provided from the PC										
CN9	SD Card Slot	Micro SD Card holder allows permanent installation for large storage										

5V pins are common un-fused input /outputs. 3V3 pins are outputs only with a total 50mA capacity. Do not connect pins <sup>1-1</sup> or NC

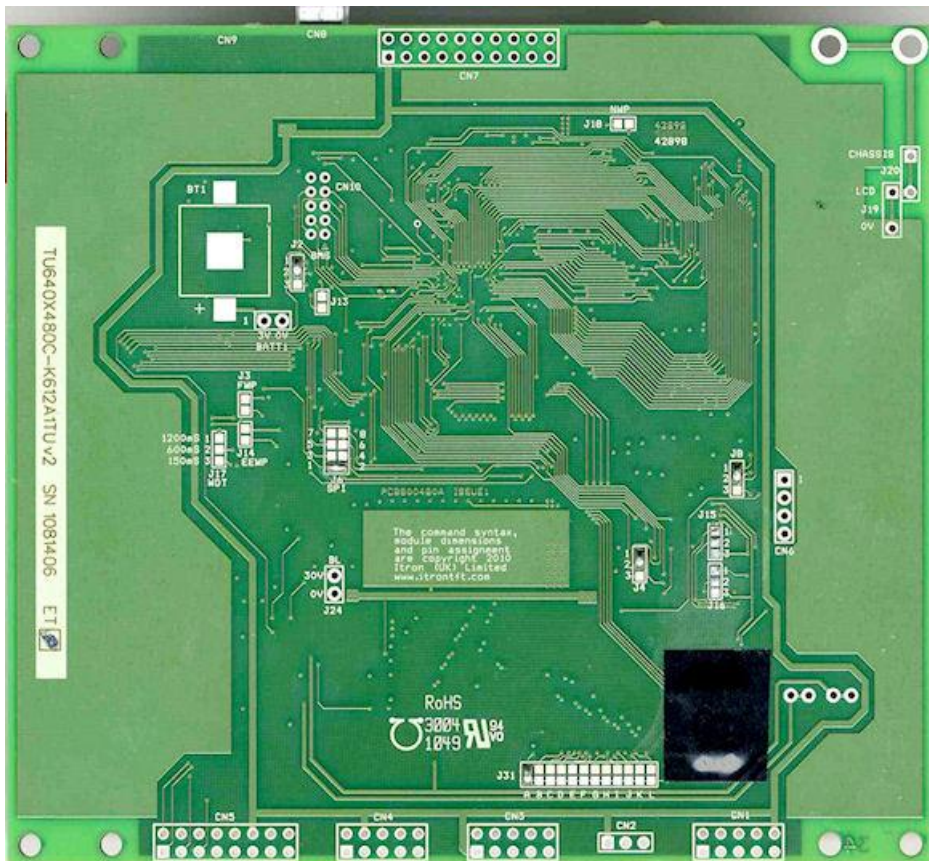
## Jumper and Additional Connector Information

JMP/CON	Function	Note
BT1	Battery Connector	Apply solder bump to center pad before fitting holder. CR2025 battery positive up.
BATT1	RTC alternate power 3VDC	Apply right angle connector top side soldered.
BL	LED Backlight alternate supply	When the backlight is software disabled, 30VDC at 20mA can be applied by the user.
J4	RTS Jumper	Solder 1 and 2 for RTS.
J15	RTS+RS4/DTR Jumper	Solder 1 and 2 for RTS and RS485 if fitted, solder 2 and 3 for DTR when RS485 not fitted.
J16	CTS+RS4/DSR Jumper	Solder 1 and 2 for CTS and RS485 if fitted, solder 2 and 3 for DSR when RS485 not fitted.
xWP	Write protect jumpers	Solder to prevent data update of non volatile memory where fitted.N=Nand, EE=EEPROM.

Note: RTS/CTS or DTR/DSR can be selected, not both. When RS485 fitted in model K611A1xx then only RTS/CTS are possible.

Rear View TU640X480-K612A1TUv2 with factory jumper setting.

iSMART Noritake Itron 5.7" TFT Module



Pin Assignments, Module Dimensions and Function Syntax Copyright 2010 Noritake Co Limited

**Accessories**

Noritake- Itron offers a range of accessories to get you up and running quickly.

**USB Cable**  
IFCKUSBminiB2M



**RS232 Cable**  
IFCK232-610A



**CAN Bus Interface**  
EMBCK33A  
Maximum speed 1MHz



**AC97 Audio Module**  
MCBK-AC97P1  
Bi-directional stereo codec and amplifier

TBA

More Details...

**Battery Holder**  
CONFSCR1216  
Uses a CR1216 battery  
Solders to rear of TFT



**USB-SD Expander**  
CBK-USBSD1  
Supplied with bezel for panel mounting

TBA

**IDC Interface Cable**  
IFCK10DC10-200A  
10 way 200mm length

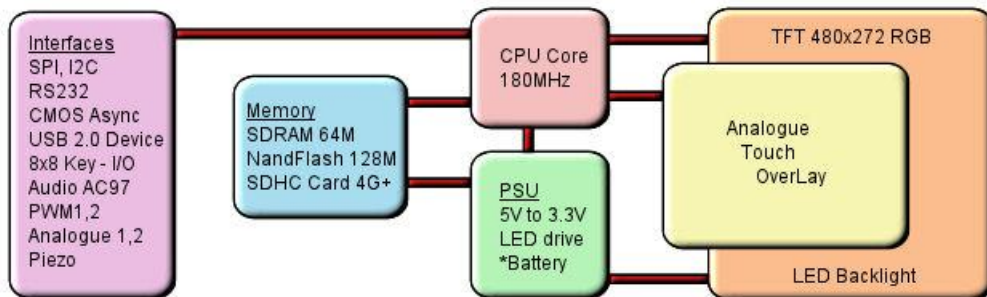


**iSMART TFT Module Overview**

**Product Overview**

This product has been designed to simplify the implementation of TFT technology into your product. The high level text based object oriented command structure, entity library and multi page screen memory allow most of the processing to be undertaken by the TFT module leaving the host CPU to concentrate on the core application processes. This allows proven firmware running on small 8 bit microcontrollers to be modified to drive this TFT with a minimum of work and risk.

**Hardware for 4.3"**



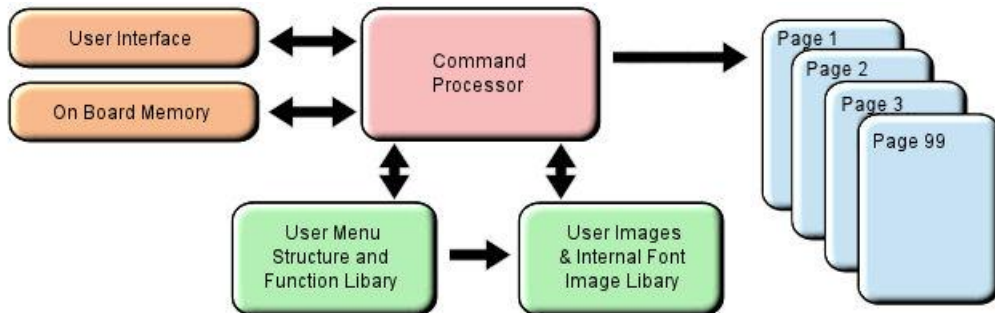
\*option

**Software Overview**

Several customers have asked why we developed our own object oriented programming language rather than provide a product with Linux or an operating system supporting compiled 'C'. If we look back at the original requirements we can see some of the reasons.

- Prime: A combined operating and communication software offering unique capabilities for slave / host applications.
- 1/ The customer's end user or distributor could write code and insert images to add in their own functionality with a text editor.
  - 2/ The program code could be updated or expanded by the host system using ASCII text over a serial link.
  - 3/ The product should be license free and use simple development tools.
  - 4/ The customer can create his own large images and control them like fonts.
  - 5/ The SD card should be able to stream video and audio with the minimum of user programming.
  - 6/ Existing host software need only have limited changes to upgrade a display from 4X20 LCD to a full colour TFT.
  - 7/ The module has the intelligence to operate as a host and the compact command language to act as a high speed slave.
  - 8/ The number of commands should be minimized by using 'overloading' and provide a higher level of functionality than C functions.
  - 9/ The parameters for interfaces and screen entities should be held in styles similar to HTML.
  - 10/ The application development time should take days or weeks rather than months.
  - 11/ If the software engineer leaves the company, it is relatively easy for the engineering manager to amend the program.

These reasons may not be key to your application, but we believe it does offer new product opportunities.



**High Level Object Oriented Commands**

The module has an integrated compiler and debugger so that users can write the high level object oriented language commands in a text file or send via an interface to develop their application. Although pictures and fonts can be loaded via an interface, it is best to store these on an SD card or transfer via USB from on a PC. The multi faceted commands are divided into 4 groups as shown below. You may be thinking how can 25 commands operate a host system, so lets take a look at the **LOAD** command. It can perform the equivalent language functions of strcpy, strcat, inp, outp and a page collation function. Please study our application example code for an understanding of this compact language.

library & system	page & visibility	draw on page	functions
<b>FPROG</b> Load Menu/Img to Flash	<b>PAGE</b> Create a page of entities	<b>POSN</b> Position cursor on page	<b>FUNC</b> Create a function
<b>LIB</b> Load Image/Font to RAM	<b>STYLE</b> Set parameters	<b>TEXT</b> draw text on page	<b>VAR</b> Create a variable
<b>INC</b> Include a sub file	<b>SHOW</b> Show a page or entity	<b>DRAW</b> draw box, circle, line, pixel	<b>IF ? :</b> Conditional test
<b>RUN</b> Call function or user code	<b>HIDE</b> Hide a page or entity	<b>IMG</b> draw image on page	<b>LOOP</b> Repeat commands
<b>RESET</b> Reset system, library, time	<b>DEL</b> Delete entity from Library	<b>KEY</b> create touch or external key	<b>CALC</b> Calculation and string edit
<b>;;</b> Refresh current page	<b>LOAD</b> Copy and format pages, strings, interface and data		<b>WAIT</b> Set delay period
<b>;</b> Terminate command			<b>INT</b> Set an interrupt

**Styles make your Application Consistent**

All entities and buffers use parameters stored in a Style similar to HTML web pages. These are extensive and define colours, entity types, buffer size and interface parameters like baud rate, clock edges and data format. Styles can be embedded in parent styles to reduce repetition and simplify changes.

**Screen Page Creation and Control**

Pages can be smaller than the screen for pop up help menus, status information and lists. Buttons can be varying size, with radio, rectangle or check box style with special types for navigation actions. The cursor position command allows relative or absolute positioning for reduced instructions during page layout. Entities can be updated by incoming host commands and their associated functions can run all the time or only when the entity or it's page is visible. When a text is numeric, it can be compared, incremented or decremented or form part of an equation using

## iSMART Noritake Itron 5.7" TFT Module

the CALC command. Buffers or variables can be created for interfaces, on-board memory, the SD Card, timers, counters and text. Hex code can be included in text variables when prefixed by \\. When creating your page structures and functions in a file, // prefixes user comments.

### Uploading your Menu Structure, Functions and Images

Data received from interfaces or flash memory is processed and stored in RAM libraries for high speed access to create or refresh pages and entities. Every entity has a text name for easy reference by future update commands.

In a similar way to a PC, your software could be permanently retained on an SD card and auto loaded at Power On or saved in internal flash by transferring it from an SD card or uploading it via an interface port. SD cards of 1G size and SDHC cards of 4G, 8G, 16G and 32G size are supported. 2G SD cards are not supported.

If an SD Card is used, the module will look for a file called 'TU480A.MNU' which will reference all other menu or image files. This may be your only menu file with all functions included. It would have a header similar to the example below to copy other files on the SD card to the internal flash memory. See the 'example projects' section

```
RESET(LIBRARY); FPROG;  
LIB(BACKIMAGE,"SDHC/backmain.bmp");           //load background picture into the onboard flash library  
LIB(STARTIMAGE,"SDHC/startbut.bmp");          //load start button into the onboard flash library  
..... FEND;
```

From Q1,2011, entities can be changed via the user interfaces by direct reference to there name or style

Examples:

```
homepage.back="BLUE";           change the background colour of the page called homepage to blue  
rs2.set="96e";                  change the rs232 baud rate to 9600 baud with even parity  
StatusText="Visual Error";     change the text area called StausText to show Visual Error  
GenText.font="40X56Kata"       change font size of all text using style GenText
```

We hope you find the 'getting started' and online examples suitable for understanding the functional techniques and rapid implementation in your application. Please do not hesitate to contact our tech team by email for assistance. [tech@noritake-itron.com](mailto:tech@noritake-itron.com)

**System Hardware Setup Parameters & Development Status - 4****System Hardware Setup Parameters and Development Status**

This product has been released to a limited market in Europe with 35 customers evaluating product prior to full release on 16th Sep 2010. This page identifies the current and expected operating status of interfaces with release dates which are subject to revision. The introduction of interface protocols (Modbus RTU) will take place in late November 2010. The parameters for an interface are defined using the command setup(Name) {...}.

Parameters	Description	Status	View
<b>asynchronous interfaces</b>	set up rs2, rs4, as1, as2, dbg		<a href="#">RS2</a>
<b>set="96NC"</b>	quick set up combination	OK	<a href="#">ASY</a>
<b>baud = num;</b>	num = 110 to 115200.	OK	
<b>data = num;</b>	num = 5, 6, 7, 8	OK	
<b>stop = num;</b>	num = 1, 1.5, 2 1.5 is 1.5 bits	OK	
<b>parity = ch;</b>	parity = Odd, Even, None, Mark, Space	OK	
<b>rx = Y or C or N;</b>	set receive buffer interface active	OK	
<b>proc = ";" or \OD or other</b>	process on receive string terminator	OK	
<b>procDel = Y or N</b>	delete or keep termination character.	OK	
<b>rx = num;</b>	set size of receive buffer in bytes.	OK	
<b>tx = Y or E or N;</b>	set transmit buffer interface	OK	
<b>tx = num;</b>	set size of transmit buffer in bytes.	OK	
<b>encode = s , w, m;</b>	s= single byte ASCII, w=2 byte UNI, m= UTF8	OK	
<b>flow = N , H, S;</b>	flow control - none, hardware, software XON XOFF	OK except XON/OFF plan 2nd Dec	

<b>spi interface</b>	set up spi , tsync, rsync	Receive OK, Transmit 26th Nov	<a href="#">SPI</a>
<b>set = "MR100";</b>	quick set up combination	OK for receive	
<b>active= M or S or N;</b>	set as Master, Slave or None	Slave Only	
<b>edge= R or F;</b>	uses Rising or Falling clock edge	OK	
<b>speed = 100;</b>	set transmit speed in master mode		
<b>rx = Y or C or N;</b>	set receive buffer interface as active	OK	
<b>proc=";" or \OD or other</b>	process on receive string terminator	OK	
<b>procDel = Y or N</b>	delete or keep termination character.	OK	
<b>encode = s , w, m;</b>	s= single byte ASCII, w=2 byte UNI, m= UTF8	OK	
<b>rx = num;</b>	set size of receive buffer in bytes	OK	
<b>rxo= M or L;</b>	set receive data order	OK	
<b>rx = N , H;</b>	use none or hardware MB		
<b>rx = N , Y;</b>	use select input \RSS.	OK	
<b>tx = Y or E or N;</b>	set transmit buffer interface as active		
<b>end= "nn"</b>	byte returned when no data left in buffer		
<b>tx = num;</b>	set size of transmit buffer in bytes.		
<b>txo= M or L;</b>	set transmit data order		
<b>tx = N , H;</b>	none or hardware HB in Master mode		
<b>tx = N , Y;</b>	use select output \TSS in master mode		

<b>i2c interface</b>	set up i2c		<a href="#">I2C</a>
<b>set = "S7E";</b>	quick set up of I2C - Slave and Address	OK	
<b>addr= "nn"</b>	address pair where nn write, nn+1 read	OK	
<b>end= "nn"</b>	byte returned when no data left in buffer	OK	
<b>active= M or S or N;</b>	set as Master Slave or None	OK	
<b>speed = 100;</b>	set transmit speed value in master mode	OK	
<b>rx = Y or C or N;</b>	set receive buffer interface as active with command	OK	
<b>proc = ";" or \OD or other</b>	process on receive string terminator	OK	
<b>procDel = Y or N</b>	delete or keep termination character.	OK	
<b>encode = s , w, m;</b>	s= single byte ASCII, w=2 byte UNI, m= UTF8	OK	
<b>rx = num;</b>	set size of receive buffer in bytes	OK	
<b>tx = Y or E or N;</b>	set transmit buffer interface as active with echo	OK	
<b>tx = num;</b>	set size of transmit buffer in bytes.	OK	

<b>key i/o interfaces</b>	K23 is the highest order bit and K0 the lowest		<a href="#">KEY</a>
<b>active</b>	high is active "\000000" > "\FFFFFF"	OK	
<b>inp</b>	high is input, low output "\000000" > "\FFFFFF"	OK	
<b>trig</b>	high is trigger interrupt	OK	
<b>edge</b>	high is rising edge, low is falling edge	OK	
<b>keyb</b>	high is scanned keyboard connection	OK	

<b>pwm controller</b>	pwm1, pwm2 - 160Hz to 1MHz	OK	<a href="#">PWM</a>
<b>active=N,1,2,12;</b>	set pwm activity None, pwem1, pwm2, pwm1 and 2		
<b>polln=H or L;</b>	poll1, poll2 is High or Low on first phase		
<b>cyklen=hhh;</b>	value in microseconds for cycle1, cycle2		
<b>dutyn=hh;</b>	value as a percentage of High period		
<b>delay=nnn;</b>	delay in microseconds between pwm1 and pwm2		

<b>analogue converters</b>	adc1, adc2 are processed at 1000 samples per second	OK	<a href="#">ADC</a>
<b>active=N,1,2,12;</b>	set none, ADC1, ADC2 or both		
<b>calib1=function name;</b>	set user function to use for calibrate/scale ADC1		
<b>calib2=function name;</b>	set user function to use for calibrate/scale ADC2		
<b>avg1= 1-16;</b>	number of samples taken and averaged for ADC1		
<b>avg2= 1-16;</b>	number of samples taken and averaged for ADC2		

<b>buzz = buzzer output</b>	Use LOAD(BUZZ,var) var=ON,OFF, or time in millisecs	OK	<a href="#">BUZ</a>
-----------------------------	---	----	---------------------

<b>other interface references</b>			
<b>internal eeprom</b>	parameter storage using extended variables VarE	OK	<a href="#">VAR</a>
<b>sdhc = SD Card (1G or 4G+)</b>	FAT32 - 8 character file names, no directory. Not 2G	Read OK. Write TBA.	<a href="#">SD</a>
<b>internal NAND flash</b>	Proprietary structure	Active v33 for firmware	
<b>usbcom = usb com port</b>		TBA	<a href="#">COM</a>
<b>usbmsd = mass storage</b>		TBA	<a href="#">MSD</a>
<b>CAN adaptor - 1MHz</b>	adaptor connects to CN3	OK	<a href="#">CAN</a>
<b>ac97= audio buffer</b>	adaptor connects to CN4	TBD	<a href="#">I2S</a>



**System, RTC and Counter Setup****System**

Set up the system. These parameters can be set at initialisation or at any time during operation by specifying the parameter to be changed. Example: `setup( system ){ bled=50; }`. To change a setting use a dot operator as follows: `LOAD(system.bled,50);`

`startup=all;`

Displaying messages and progress bar at start up is configurable using `startup=all` or `none` or `bar`.

`bled = 100;`

set backlight to OFF=0 or ON=100 (1-99 brightness levels available v4 PCB, v32 firmware)

`wdog = 1000;`

set the watchdog time out period in milliseconds.

`rotate = 0;`

set the rotation of the screen with respect to PCB. This is stored in EEPROM for use with boot messages.

`test=hide/showTouchAreas;`

hide or show touch areas during product development

`encode = s, w, m;`

menu text strings can contain single byte ASCII (s), 2 bytes for Unicode (w) or multibyte for UTF8 (m)

`calibrate = n;`

initialise the internal touch screen calibration screen. This automatically returns to the previous page on completion. If it is necessary to abort then send `setup( system ) {calibrate=n};`

`touchsamples = 20;`

Define the number of touch samples per interrupt. Defaults: 4.3" = 12; 5.7" = 12; 7" = 22;

`touchdebounce = 10;`

Define the time period between each sampling period. Defaults: 4.3" = 25; 5.7" = 30; 7" = 25;

`touchaccuracy = 20;`

Define the 0.25 pixel accuracy of the samples. Defaults: 4.3" = 50; 5.7" = 14; 7" = 12;

**Example system set up**

```
setup(system)
{
  bled=100;
  wdog=100;
  rotate=0;
  calibrate=n;
  test=showTouchAreas;
  encode=s; //ASCII handling with extended unicode/utf8 in occasional strings
  touchsamples = 10;
  touchdebounce = 5; //picture drag and drop settings to emulate capacitive keyboard
  touchaccuracy = 30;
}
```

**system version**

The software and hardware versions can now be read to a serial port or text variable.

```
LOAD( RS2, VERS_IBOOT ) returns NAND bootloader version
LOAD( RS2, VERS_ILOADER ) returns main loader version
LOAD( RS2, VERS_IAPP ) returns main application version
LOAD( RS2, VERS_IMODULE ) returns module name and version
```

**operational****Real Time Clock RTC**

The real time clock requires a battery to be fitted to the rear of the module or a 3VDC supply applied via a connector fitted to the rear of the PCB. The default format is 14 Sep 2010 09:50:06 which can be modified to suit the application which is achieved by loading the RTC into a variable having the required format. Another method is to use predefined variables of individual RTC values.

**SET RTC**

The RTC is set using 24 hour time with `LOAD( RTC, "YYYY:MM:DD:hh:mm:ss" );`

with fixed format where:

- YYYY is year 1900-2099
- MM is month 01-12
- DD is day of month 01-31
- hh is hours 00-23
- mm is minutes 00-59
- ss is seconds 00-59

Use vars to setup the time in a user page

```
VAR(years,2010,U16);
VAR(months,11,U8);
VAR(days,2,U8);
VAR(hours,10,U8);
VAR(mins,30,U8);
```

User changes the vars via buttons then a SAVE button would load the RTC

```
LOAD(RTC,years,":",months,":",days,":",hours,":",mins,":00");
```

**READ RTC**

You can LOAD the RTC into a variable where the format is specified in a style as follows:

```
STYLE( myRtcStyle, Data )
{
  type = text; // Setup a text variable
  length = 64; // with max length of 64 chars
  format = "jS F Y g:ia"; // RTC format string
}
```

```
VAR( RtcVar, "", myRtcStyle ); // Create a var to store formatted string
LOAD( RtcVar, RTC ); // Grab the formatted RTC time and date
TEXT( Txt1, RtcVar ); // Show the formatted time on display in Txt1 and refresh screen
LOAD( RS2, RtcVar ); // Send formatted time on RS232 port
```

The RTC date/time can be displayed as a formatted string using special characters

```
> Day:
  d Day of month with leading zeros          01-31
  j Day of month without leading zeros       1-31
  S Ordinal suffix for day of month         st, nd, rd, th

> Month:
  F Full textual representation of month     January-December
  m Numeric representation of month with leading zeros 01-12
  M Short textual representation of month, three letters Jan-Dec
  n Numeric representation of month without leading zeros 1-12

> Year:
```

# iSMART Noritake Itron 5.7" TFT Module

Y	Full numeric representation of year, 4 digits	1900-2099
y	Two digit representation of year	00-99
> Time:		
a	Lowercase Ante meridiem and Post meridiem	am, pm
A	Uppercase Ante meridiem and Post meridiem	AM, PM
g	12-hour format of hour without leading zeros	1-12
G	24-hour format of hour without leading zeros	0-23
h	12-hour format of hour with leading zeros	01-12
H	24-hour format of hour with leading zeros	00-23
i	Minutes with leading zeros	00-59
s	Seconds with leading zeros	00-59
> other characters not in list will be shown as is		

**Format examples:**

"d M Y H:i:s" will display as: 14 Sep 2010 09:50:06 (default format)  
 "d/m/y" will display as: 14/09/10  
 "jS F Y g:ia" will display as: 14th September 2010 9:50am

Predefined variables below can be read, but not set.  
 numeric variable containing seconds (0-59) which can be tested or loaded into a text.  
 numeric variable containing minutes (0-59) which can be tested or loaded into a text.  
 numeric variable containing hours (0-23) which can be tested or loaded into a text.  
 numeric variable containing days (1-31) which can be tested or loaded into a text.  
 numeric variable containing month (1-12) which can be tested or loaded into a text.  
 numeric variable containing year (1900-2099) which can be tested or loaded into a text.

**operational**

---

**Runtime Counter**

The RUNTIME counter uses pre-define variables which can be set and tested for values  
 The command Reset(RUNTIME) sets all vales to zero and starts the timer.  
 This runtime counter is independent of the real time clock and runs continually so no setup is required.

CNTMILLI	Increments every millisecond 0-999
CNTSECS	Increments every second 0-59
CNTMINS	Increments every minute 0-59
CNTHOURS	Increments every hour 0-23
CNTDAYS	Increments every 24 hours

Example Usage

IF(CNTMINS>30,FuncHalfHour); //if greater than 30 minutes run function called FuncHalfHour  
 TEXT(MinsText,CNTMINS); //update counter value on page and refresh screen

**operational**

---

**I/O Counters**

The 24 I/O counters use pre-define variables which can be reset and tested for value.  
 The counter uses an unsigned 32bit register (U32) with names CNTKxx where xx=00 to 23.  
 They require the I/O to be set as an interrupt but **do not** require an associated INT() command.  
 Counter increment depends on the rising or falling edge of the interrupt.  
 The command RESET(CNTK00) resets to zero the I/O counter on K00.  
 The maximum counter speed is 0-10kHz+ and is dependent on other interrupt and entity usage.

CNTK00	Counter on I/O K00 (CN7)
CNTK01	Counter on I/O K01 (CN7)
CNT22	Counter on I/O K22 (CN4)
CNT23	Counter on I/O K23 (CN4)

Example Usage

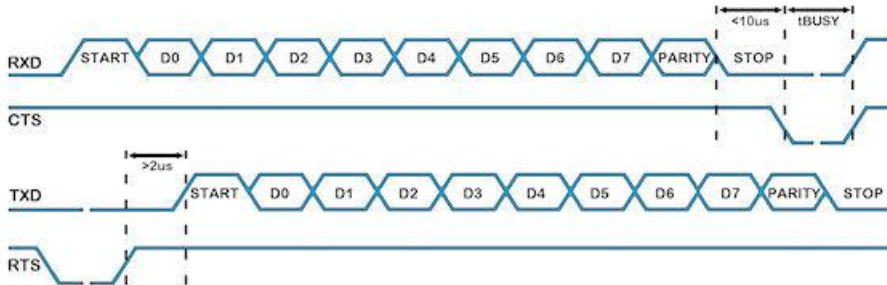
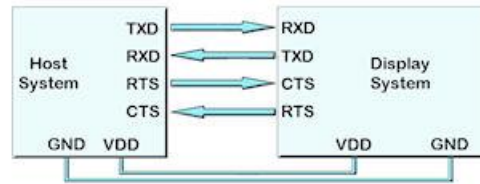
IF(CNTK00>300,Func300); //if greater than 300 run function called Func300  
 TEXT(K00Text,CNTK00); //update counter value on page and refresh screen

**operational v40**

**RS232 Interface - RS2**

The asynchronous communication speed and parity can be set with the setup command. The hardware lines RTS-CTS and DTR-DSR enable communication between host and module and are selected by jumpers on the back of the module. Only one pair can be selected at any one time. (RTS-CTS or DTR-DSR).

If RS485 is available on the module (suffix -K611xxx) then only RTS-CTS can be used.

**rs232 set up parameters**

```

setup(RS2)
{
  set="96NC";      //quick set up combination "48, 96, 192, 384, 768, 1150 with parity N, O, E and Command option".
}

setup(RS2)
{
  //user must test the application to establish the maximum viable baud rate.
  baud=38450;      //num = 110 to 6,250,000. Any value can be set to allow trimming for deviating clocks i.e. 38450
  data=6;         //num = 5, 6, 7, 8
  stop=15;        //num = 1, 15, 2 - note 15 is 1.5 bits
  parity=N;       //first letter of Odd, Even, None, Mark, Space
  rxi=Y;          //set receive interface as active (Y), a command processing source (C) or disable (N). Default = N
  proc=",";       //process on receive termination character. See below
  procDel=Y;      //remove or keep the termination character(s) before processing
  rxb=8246;       //set size of receive buffer in bytes. Default = 8192 bytes maximum = 256K bytes.
  txi=Y;          //set transmit interface as active (Y), to echo command processing (E) or disable (N)
  txb=8350;       //set size of transmit buffer in bytes. Default = 8192 bytes
  encode=s;       //set s=ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw data bytes.
  flow=N;         //none (N), hardware RTS/CTS or DTR/DSR (H), software XON XOFF (S).
}

```

**Serial Port Interrupt Characters**

Serial Port termination characters can now be specified to generate an interrupt. The proc parameter is used in the port setup to define the termination character(s).

```

proc = all;      <- trigger on all received characters
proc = CRLF;    <- trigger on a CR followed by LF (0Dh 0Ah)
proc = CR;     <- trigger on CR (0Dh) ...in command mode rxi=C this is fixed
proc = LF;     <- trigger on LF (0Ah)
proc = NUL;    <- trigger on NUL (00h)
proc = \\xx;   <- trigger on xxh
proc = "ABCD"; <- string in format defined by SYSTEM encode param
proc = "\\xx\\xx"; <- string in format defined by SYSTEM encode param

```

When sending commands (rxi=C) to the module, processing only occurs when \\0D or 0D hex is received. Example: TEXT(MyText,"Hello World");\\0D

Parameter can be updated using the dot operator  
LOAD(RS2.baud,19200);  
LOAD(RS2.proc,"CR");

**Example usage**

```

setup(RS2)
{
  set="96NC"      //quick set up combination "48, 96, 192, 384, 768, 1150 with parity N, O, E and Command option".
}

PAGE( PageName, PageStyle)
{
  POSN(100,100); TEXT ( RecvTxt, "Example", stRecvTxt); //show received ASCII data on screen
  ....
  ....
  INT( SerRxInt, RS2RXC, SerRxEvent ); //Used when rxi=Y
}

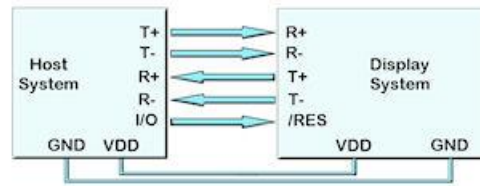
FUNC( SerRxEvent )
{
  LOAD( Var, RS2 ); // Must read RS2 to clear interrupt
  TEXT ( RecvTxt, Var); //show received ASCII data on screen and refresh. To update, no style is specified.
}

```

**Active v22 except flow control**

**RS485 Interface - RS4**

RS485 is available on the module (suffix -K611xxx)  
The asynchronous communication speed and parity can be set with the setup command.

**rs485 set up parameters**

```

setup(RS4)
{
  set="96NC";           //quick set up combination "48, 96, 192, 384, 768, 1150 with parity N, O, E and Command option".
}

setup(RS4)
{ //user must test the application to establish the maximum viable baud rate.
  baud=38450;           //num = 110 to 6,250,000. Any value can be set to allow trimming for deviating clocks i.e. 38450
  data=6;               //num = 5, 6, 7, 8
  stop=15;              //num = 1, 15, 2 - note 15 is 1.5 bits
  parity=N;             //first letter of Odd, Even, None, Mark, Space
  rxi=Y;                //set receive interface as active (Y), a command processing source (C) or disable (N). Default = N
  proc=",";             //process on receive termination character(s). See below
  procDel=Y;            //remove or keep the termination character(s) before processing
  rxb=8196;             //set size of receive buffer in bytes. Default = 8192 bytes, maximum 256K bytes.
  txi=Y;                //set transmit interface as active (Y), to echo command processing (E) or disable (N)
  txb=8196;             //set size of transmit buffer in bytes. Default = 8192 bytes
  encode=s;             //set s=ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw data bytes.
  flow=N;               //none (N), software XON XOFF (S).
  duplex=F;            //set Full Duplex (F) or Half Duplex (H). TBA
}

```

**Serial Port Interrupt Characters**

Serial Port termination characters can now be specified to generate an interrupt.  
The proc parameter is used in the port setup to define the termination character(s).

```

proc = all;             <- trigger on all received characters
proc = CRLF;           <- trigger on a CR followed by LF (0Dh 0Ah)
proc = CR;             <- trigger on CR (0Dh) ...in command mode rxi=C this is fixed
proc = LF;             <- trigger on LF (0Ah)
proc = NUL;           <- trigger on NUL (00h)
proc = \xx;           <- trigger on xxh
proc = "ABCD";        <- string in format defined by SYSTEM encode param
proc = "\xx\yy";      <- string in format defined by SYSTEM encode param

```

When sending commands (rxi=C) to the module, processing only occurs when \0D or 0D hex is received.  
Example: TEXT(MyText,"Hello World");\0D

Parameter can be updated using the dot operator  
LOAD(RS4.baud,19200);  
LOAD(RS4.proc,"CR");

**Example usage**

```

setup(RS4)
{
  set="96NC"           //quick set up combination "48, 96, 192, 384, 768, 1150 with parity N, O, E and Command option".
}
PAGE( PageName, PageStyle)
{
  POSN(100,100); TEXT ( RecvTxt, "Example", stRecvTxt); //show received ASCII data on screen
  ....
  ....
  INT( SerRxInt, RS4RXC, SerRxEvent ); //Used when rxi=Y
}

FUNC( SerRxEvent )
{
  LOAD( Var, RS4 ); // Must read RS4 to clear interrupt
  TEXT ( RecvTxt, Var); //show received ASCII data on screen and refresh. To update, no style is specified.
}

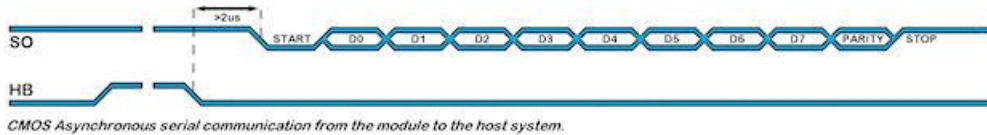
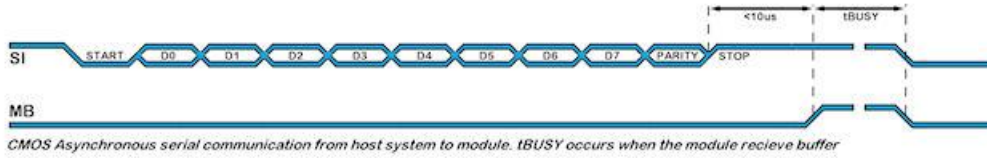
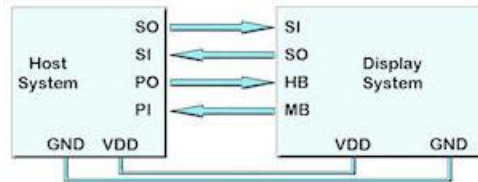
```

**Operational except half duplex**

**CMOS Asynchronous Interface - 8**

**CMOS Asynchronous Interfaces - AS1, AS2, DBG (3v3 level)**

The asynchronous communication speed and parity can be set with the setup commands. The host busy line (HB) stops the module from sending data to the host. The use of the HB and MB busy lines are optional, and can be connected together if not required.



**AS1, AS2, DBG set up parameters**

```

setup(AS1)           //can setup AS1, AS2 or DBG
{
  set="96NC";        //quick set up combination "48, 96, 192, 384, 768, 1150 with parity N, O, E and Command option".
}

setup(AS1)           //can setup AS1, AS2 or DBG
{
  //user must test the application to establish the maximum viable baud rate.
  baud=38450;         //num = 110 to 6,250,000. Any value can be set to allow trimming for deviating clocks i.e. 38450
  data=7;             //num = 5, 6, 7, 8
  stop=2;             //num = 1, 15, 2 - note 15 is 1.5 bits
  parity=N;           //first letter of Odd, Even, None, Mark, Space
  rxi=Y;              //set receive buffer interface as active (Y), a command processing source (C) or disable (N). Default = N
  proc=",";           //process on receive termination character(s). See below
  procDel=Y;          //remove or keep the termination character(s) before processing
  rxb=8246;           //set size of receive buffer in bytes. Default = 8192 bytes, maximum 256K bytes.
  txi=Y;              //set transmit buffer interface as active (Y), to echo command processing (E) or disable (N)
  txb=8246;           //set size of transmit buffer in bytes. Default = 8192 bytes
  encode=s;           //set s=ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw data bytes.
  flow=N;             //none (N), hardware RTS/CTS or DTR/DSR (H), software XON XOFF (S).
}
    
```

**Serial Port Interrupt Characters**

Serial Port termination characters can now be specified to generate an interrupt.

The proc parameter is used in the port setup to define the termination character(s).

```

proc = all;          <- trigger on all received characters
proc = CRLF;        <- trigger on a CR followed by LF (0Dh 0Ah)
proc = CR;          <- trigger on CR (0Dh) ...in command mode rxi=C this is fixed
proc = LF;          <- trigger on LF (0Ah)
proc = NUL;         <- trigger on NUL (00h)
proc = \xx;         <- trigger on xxh
proc = "ABCD";      <- string in format defined by SYSTEM encode param
proc = "\xx\yy";    <- string in format defined by SYSTEM encode param
    
```

When sending commands (rxi=C) to the module, processing only occurs when \0D or 0D hex is received.

Example: TEXT(MyText,"Hello World");\0D

Parameter can be updated using the dot operator

```

LOAD(AS1.baud,19200); //can load AS1, AS2 or DBG
LOAD(AS1.proc,"CR");  //can load AS1, AS2 or DBG
    
```

**Example**

```

setup(AS1)           //can setup AS1, AS2 or DBG
{
  set="96NC"         //quick set up combination "48, 96, 192, 384, 768, 1150 with parity N, O, E and Command option".
}

PAGE( PageName, PageStyle)
{
  POSN(100,100); TEXT ( RecvTxt, "Example", stRecvTxt); //show received ASCII data on screen
  ....
  ....
  INT( ASerRxInt, AS1RXC, SerRxEvent ); //Used when rxi=Y
}

FUNC( SerRxEvent )
{
  LOAD( Var, AS1 ); // Must read AS1 to clear interrupt
  TEXT ( RecvTxt, Var); //show received ASCII data on screen and refresh. To update, no style is specified.
}
    
```

# iSMART Noritake Itron 5.7" TFT Module

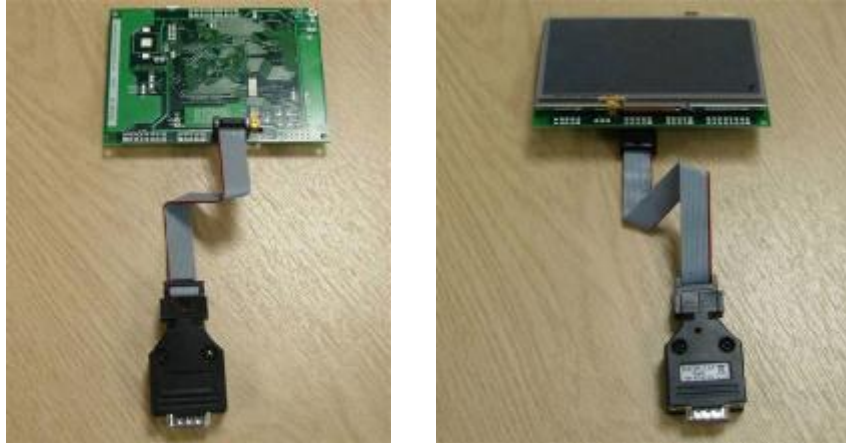
## CANBUS Adaptor

When attaching a CANBUS adaptor type EMCBK33 to CN3 using a 10 way IDC cable, the connector is fitted to the backside of the module and the following set up is required to match the default settings in the adaptor.

```
setup(AS1)
{
  baud=38400; //num = 110 to 115200. Any value can be set to allow trimming for deviating clocks i.e. 38450
  data=8; //num = 5, 6, 7, 8
  stop=1; //num = 1, 1.5, 2 - note 1.5 is 1.5 bits
  parity=N; //first letter of Odd, Even, None, Mark, Space
  rxi=C; //set receive buffer interface as active (Y), a command processing source (C) or disable (N). Default = N
  encode=sr; //set s=ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw data bytes.
  flow=H; //none, hardware RTS/CTS or DTR/DSR, software XON XOFF
}
```

The default receive address for the adaptor is ID=155h with 11bit or 29bitID packets accepted (2.0a or 2.0b spec)  
All bytes are received on AS1 with 1 to 8 bytes of data.  
The transmit ID is also 155H. with data sent via AS1 with data length of 1.

Connection to an iSMART TFT is shown below.

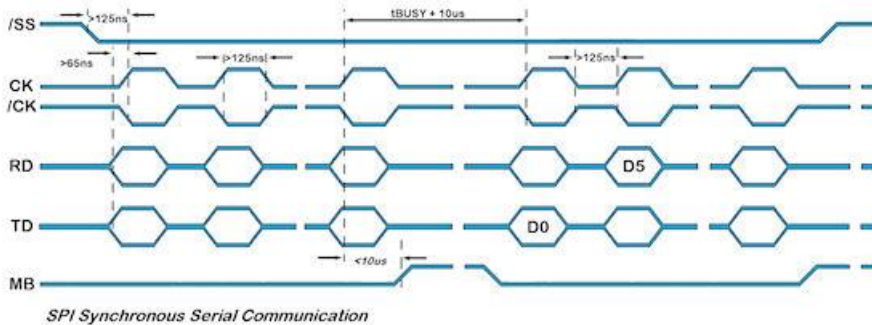
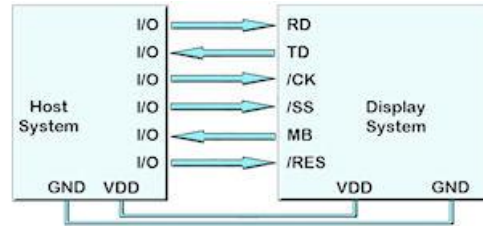


**Active**

**SPI Interface****SPI Interface - SPI (3v3 level)**

With synchronous communications enabled, data can be clocked into the TFT module using the rising or falling edge of SCK. This is selectable by the setup command which also sets other parameters. By default, data is clocked in on the rising edge with the most significant bit sent first.

The /SS pin can be used as an enable pin if other devices are connected to the serial line and also allows byte synchronization. If MB is set high, the input buffer is full or disabled. A dummy/end byte for reading and buffer status can be set by the user.

**spi - set up parameters**

```

setup(spi)
{
  set="MR100";           //quick set up as Master/Slave, edge R/F, Command and speed 20-1000
}

setup(spi)
{
  active=M;              //set as Master, Slave or None for both transmit and receive. Default = N
  edge=R;                //uses Rising or Falling clock edge. Default = R
  speed=100;             //set transmit speed value in kilobits/sec from 20 to 1000 for master mode. Default = 100
  rxi=Y;                 //set receive buffer interface as active (Y), a command processing source (C) or disable (N). Default = N
  proc=",";              //process on receive termination character(s). See below.
  procDel=Y;            //remove or keep the termination character(s) before processing
  encode=s;              //set s=ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw data bytes.
  rxb= 8264;             //set size of receive buffer in bytes. Default = 8192 bytes
  rxo=M;                 //set receive data order as most significant bit (M) or least significant bit (L). Default = M
  rxf= N;                //use none or hardware MB to signify receive buffer full. Default = N
  rxs=N;                 //use select input \RSS. Default = N
  txi=Y;                 //set transmit buffer interface as active (Y), to echo command processing (E) or disable (N)
  end="nn"               //byte returned when no data left in display's spi transmit buffer and as a dummy byte to send if required.
  txb=8244;              //set size of transmit buffer in bytes. Default = 8192 bytes
  txo=M;                 //set transmit data order as most significant bit (M) or least significant bit (L). Default = M
  txf=N;                 //none or hardware HB used to signify halt transmit in master mode. Default = N
  txs=N;                 //use select output \TSS in master mode. Default = N
}

```

**Serial Port Interrupt Characters**

Serial Port termination characters can now be specified to generate an interrupt. The proc parameter is used in the port setup to define the termination character(s).

```

proc = all;              <- trigger on all received characters
proc = CRLF;            <- trigger on a CR followed by LF (0Dh 0Ah)
proc = CR;              <- trigger on CR (0Dh) ...in command mode rxi=C this is fixed
proc = LF;              <- trigger on LF (0Ah)
proc = NUL;             <- trigger on NUL (00h)
proc = \xx;             <- trigger on xxh
proc = "ABCD";          <- string in format defined by SYSTEM encode param
proc = "\xx\yy";        <- string in format defined by SYSTEM encode param

```

When sending commands (rxi=C) to the module, processing only occurs when \0D or 0D hex is received. Example: TEXT(MyText,"Hello World");\0D

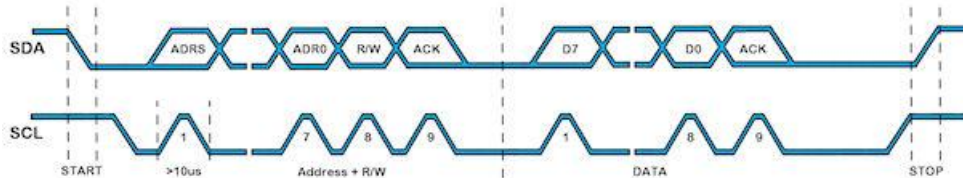
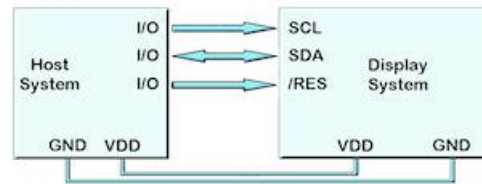
Parameter can be updated using the dot operator  
LOAD(spi.baud,19200);  
LOAD(spi.proc,"CR");

**SPI receive active v32. Master and transmit plan 6th Feb**

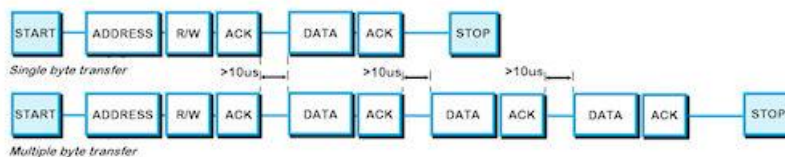
**I2C Interface****TWI / I2C Interface - I2C (3v3 level)**

The I2C interface operates as a slave either in 'slave receive' or 'slave transmit' mode with a user defined address set in the I2C setup. Receive (i2c.rxb) and transmit (i2c.txb) buffers of 8192 bytes are created which can be cleared and set by the command processor. An end byte indicating empty buffer can be set.

The user must fit 10K pull up resistors to SDA and SCL somewhere on their I2C bus.



Typical I2C transmission

**An overview of how TWI / I2C communicates**

A START condition is signalled by driving SDA low while SCL is high. A STOP condition is signalled by driving SDA high while SCL is high. After a START condition is detected followed by address + R/W bit, the command / data bytes are stored in a 8192 byte buffer. The module will pull SDA low during the 9th clock cycle of a data transfer to acknowledge the receipt of a byte. Additional data may be sent providing the host receives an Ack. If the host has not detected an Ack the data transfer must be started again by providing a STOP and START condition and address + R/W bit low. When reading an I2C packet must be sent with address+1 read the data bytes from the I2C transmit buffer.

**twi / i2c set up parameters**

```

setup(i2c)
{
  set = "C7E";           //quick set up of I2C - Slave with Command and Address
}

setup(i2c)
{
  addr="3E";             //address pair where nn for write and nn+1 for read with range 02 to FE.
  end="\00";             //byte returned when no data left in display's i2c transmit buffer
  active=S;              //set as Master (M) or Slave (S) or disabled (N). Default = N
  speed=100;             //set transmit speed value in kilobits/sec from 20 to 400 for master mode. Default = 100
  rxi=Y;                 //set receive buffer interface as active (Y), a command processing source (C) or disable (N). Default = N
  proc=",";              //process on receive termination character(s)
  procDel=Y;             //remove or keep the termination character(s) before processing
  encode=s;              //s= ASCII single byte, w=UNICODE 2 byte, m=UTF8 multibyte
  rxb=8192;              //set size of receive buffer in bytes. Default = 8192 bytes
  txi=Y;                 //set transmit buffer interface as active (Y), to echo command processing (E) or disable (N)
  txb=8186;              //set size of transmit buffer in bytes. Default = 8192 bytes
}

```

**Serial Port Interrupt Characters**

Serial Port termination characters can now be specified to generate an interrupt.

The proc parameter is used in the port setup to define the termination character(s).

```

proc = all;              <- trigger on all received characters
proc = CRLF;            <- trigger on a CR followed by LF (0Dh 0Ah)
proc = CR;              <- trigger on CR (0Dh) .. when sending commands (rxi=C), this is fixed
proc = LF;              <- trigger on LF (0Ah)
proc = NUL;             <- trigger on NUL (00h)
proc = \xx;            <- trigger on xxh
proc = "ABCD";         <- string in format defined by SYSTEM encode param
proc = "\xx\yy";       <- string in format defined by SYSTEM encode param

```

When sending commands (rxi=C) to the module, processing only occurs when \0D or 0D hex is received.

Example: TEXT(MyText,"Hello World");\0D

Parameter can be updated using the dot operator

```

LOAD(i2c.baud,19200);
LOAD(i2c.proc,"CR");

```

Please view the I2C Master Mode example project from which this section is taken.

```

SETUP( I2C ) //master mode setup
{
  active = M;
  end = \00; //necessary to choose a character for end of string
  speed = 100;
  encode = sr; //use raw data
  rxi = Y;
  txi = Y;
}

```



## iSMART Noritake Itron 5.7" TFT Module

```
VAR(null,0,U8);  
// measure temperature using I2C sensor which has 40ms processing time  
// the 2nd byte of the load command defines the device base address. The iSMART adjusts this depending on read or write instruction.  
// the 3rd byte defines the number of bytes to read after commands (4th+ bytes) are sent.  
  
LOOP{readTempLoop,forever} {  
  LOAD(I2C,addr_temp,null,0); //addr_temp variable has \\72 for temperature sensor I2C address. Command 0 is sent with no bytes read.  
  WAIT(40);  
  LOAD(I2C,addr_temp,2); // read 2 bytes of data into I2C buffer  
  WAIT(2);  
  LOAD(temp_high, I2C); // each byte is read one at a time since raw data (encode=sr;) is defined in setup.  
  LOAD(temp_low, I2C);  
  IF(tuvar=1?convertt); //the function convertt is used to combine the 2 bytes and show degrees C or F according to user setting  
  TEXT(tempval, temp_high);; //update textbox and refresh screen  
}  
Operational
```

**Keyboard and I/O interface - 11**

**Keyboard and I/O Interfacing + PWM, ADC and Piezo**

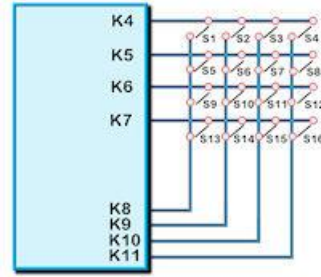
**Keyboard Control**

24 I/O lines (K0-K23) can be configured to scan a key matrix with up to 144 keys configured using the setup commands for I/O control. When a key is pressed, a function can be initiated using a key command.

Dual key presses are supported to enable SHIFT functionality.

No diodes are required in the key matrix for dual key operation making it ideal for low cost membrane keyboards.

NOTE: The KEY() function requires Kn connects to Km.  
To use Kn connects to GND, use an INT(Name,Kn,function); command



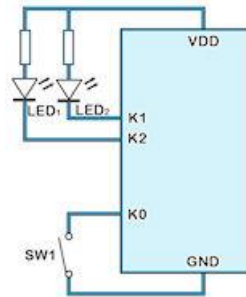
**I/O Control**

The module contains simple Input and Output functions for the 24 I/O lines (K0-K23). All inputs include an optional pull-up resistor ~50K-120K in value. The outputs can source ~1mA and sink ~3mA. Certain I/O have expanded functions for customization.

NOTE: The ports K0 to K15 connect directly to the CPU without ESD protection. K16 to K23 have series 100R resistors and 10pF capacitors to GND.

K23 is the highest order bit and K0 the lowest.

NOTE: To use Kn connects to GND, use an INT(Name,Kn,function); command



**keyio                      K00-K23                      24 bits of user i/o and keyboard                      operational**

```

setup(keyio)
{
  active=\\0000FF;           //high is active "\\000000" >"\\FFFFFF", default is inactive
  inp=\\00000C;             //high is input, low is output "\\000000" >"\\FFFFFF"
  trig=\\000001;           //high is trigger interrupt "\\000000" >"\\FFFFFF" as defined by edge
  edge=\\000000;           //high is rising edge, low is falling edge "\\000000" >"\\FFFFFF"
  keyb=\\000FF0;           //high is scanned keyboard connection "\\000000">"\\FFFFFF"
}
    
```

Single bit variables can be set and tested K00, K01, K02...K23 once enabled  
 8 bit variables can be set and tested KA, KB, KC, KD, KE once enabled  
 KA = K07,K06,K05,K04,K03,K02,K01,K00  
 KB = K15,K14,K13,K12,K11,K10,K09,K08  
 KC = K14,K12,K10,K08,K06,K04,K02,K00  
 KD = K15,K13,K11,K09,K07,K05,K03,K01  
 KE = K23,K22,K21,K20,K19,K18,K17,K16

**example usage to set**

```

LOAD(K01,1); set K1 to high
LOAD(K02,0); set K2 to low
LOAD(KA,\\02); set K0,K2-K7 low and K1 high

LOAD(myVar,K01) load port into user variable
LOAD(myVar,KA) load 8bit port into user variable
    
```

**example usage with interrupt**

```

SETUP(keyio)
{
  active=\\000001;         // set K00 to be active
  inp=\\000001;           // set K00 as input
  trig=\\000001;         // enable trigger interrupt on K00
  edge=\\000000;         // set to trigger in falling edge
}

PAGE(mypage,pagestyle)
{
  //set up entities or keys on page
  INT(myInt,K00,myEvent); // setup interrupt to call 'myEvent' on every K00 event
  //rest of page
}

FUNC(myEvent) // This function is called each time a falling edge is detected on K00
{
  // some actions
}
    
```

The current firmware requires the K parameter to be 3 characters in length

**I/O counters CNTK00-CNTK23**

The 24 I/O counters use pre-define variables which can be reset and tested for value. The counter uses an unsigned 32bit register (U32) with names CNTKxx where xx=00 to 23. They require the I/O to be set as an interrupt but **do not** require an associated INT() command. Counter increment depends on the rising or falling edge of the interrupt. The command RESET(CNTK00) resets to zero the I/O counter on K00. The maximum counter speed is 0-10kHz+ but is dependent on other interrupt and entity usage.

CNTK00	Counter on I/O K00 (CN7)
CNTK01	Counter on I/O K01 (CN7)

# iSMART Noritake Itron 5.7" TFT Module

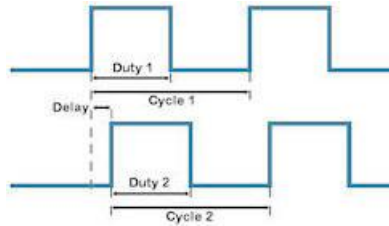
CNT22 Counter on I/O K22 (CN4)  
 CNT23 Counter on I/O K23 (CN4)

Example Usage IF(CNTK00>300,Func300); //if greater than 300 run function called Func300  
 TEXT(K00Text,CNTK00); //update counter value on page and refresh screen  
**operational v40**

## pwm controller PWM1,PWM2 **operational**

```

setup(pwm)
{
    active=12;           //use 12 to synchronize PWM 1 and 2. N=none
    pol1=H;             //polarity = High or Low on first phase of PWM1
    pol2=H;             //polarity = High or Low on first phase of PWM2
    cycle1="200";       //cycle time in microseconds of PWM1. Range 160Hz to 1MHz
    cycle2 = "300";     //cycle time in microseconds of PWM2. Range 160Hz to 1MHz
    duty1= "44";        //value of first phase as a percentage for PWM1 = 1-99
    duty2= "56";        //value of first phase as a percentage for PWM2 = 1-99
    delay= "50";        //delay between first phase of PWM1 and first phase of PWM2 in microseconds
}
    
```

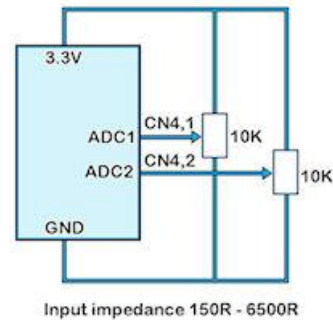


## a to d converters ADC1, ADC2 **operational**

The input voltage range is 0V to 3VDC max. Ref voltage is filtered from 3.3VDC.  
 ADC1 and ADC2 are sampled each 1ms using 10bit successive approximation.  
 If the result is copied to an 8 bit variable, the high order bits are used.

```

setup( adc )
{
    active=12;           //set none, ADC1, ADC2 or both
    calib1=0.4;         //set value to use for calibration/scaling of ADC1
    calib2=0.2;         //set value to use for calibration/scaling of ADC2
    avg1=16;            //number of samples read and then averaged for ADC1
    avg2=16;            //number of samples read and then averaged for ADC2
}
    
```



### example usage

```

//TU480A.MNU Menu file for TU480X272C with single red pen.
STYLE(BlackPg, Page) { Back=\00FF66; } //green background
STYLE(stGraphRed,DRAW){type=graph; col=red; width=4; maxX=490; maxY=300; curRel=CC; } //red pen for graph
SETUP( adc ){active=1; calib1=0.2; avg1=8; }
    
```

```

VAR(varADC1,0,U16);
VAR(PixXVal,1,U16);
    
```

```

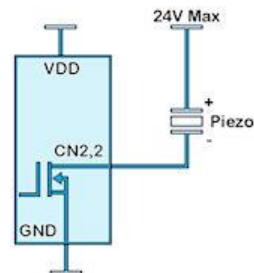
Page(GraphPage,BlackPg)
{
    POSN(240,136);
    DRAW(MyGraphRed,480,272,stGraphRed);
    LOOP(GraphLoop,FOREVER)
    {
        LOAD(varADC1,ADC1);
        DRAW(MyGraphRed,PixXVal,varADC1);
        IF(PixXVal>478?[LOAD(PixXVal,1);RESET(MyGraphRed)];:[CALC(PixXVal,PixXVal,4,"+");]); //Move to next X Pixel
    }
}
SHOW(GraphPage);
    
```

## piezo - BUZZ

CN2 is a pin connector where the centre pin (2) is connected to a 30V FET switching to 0V with maximum 200mA.  
 Users can attach a piezo sounder with integrated oscillator or similar low ripple device to provide an audible output or drive an LED indicator.  
 The negative terminal of the device should be connected to the TFT and the positive to a supply from 5V to 24VDC.

```

Use the reserved interface word BUZZ to control the output.
LOAD(BUZZ,ON);
LOAD(BUZZ,OFF);
LOAD(BUZZ,500); //sounds for 500ms - half a second.
LOAD(BUZZ,varBuzz); // varBuzz is a user declared variable with a duration variable.
    
```



## Command Overview and Development Status

This page identifies the current and expected operating status of commands and styles

Click command view column for detailed description and check release dates which are subject to revision.

The commands have a YELLOW background and the styles a PURPLE background.

Command, Style, Variable	Description	Status	View
<b>FPROG.....FEND</b>	Store menu and image files in onboard flash	Plan	<a href="#">FPROG</a>
<b>LIB(Name,Source)</b>	Load picture or font into library	OK	<a href="#">LIB</a>
<b>INC(FileName)</b>	Include the contents of another menu, style or setup file	OK	<a href="#">INC</a>
<b>RUN(Func)</b>	Run a function or user code	OK except custom code	<a href="#">RUN</a>
<b>RESET(Name)</b>	Clear eeprom variables, delete list, library or reset system	OK except library and deleted	<a href="#">RESET</a>
<b>LOAD(Name,N2,N3,N..)</b>	Multi function copy pages, variable N2--N.. to Name.	OK	<a href="#">LOAD</a>
<b>SHOW(Name)</b>	Show a page, entity	OK	<a href="#">SHOW</a>
<b>HIDE(Name)</b>	Hide a page, entity	OK	<a href="#">HIDE</a>
<b>DEL(Name)</b>	Delete a page, entity	OK	<a href="#">DEL</a>
<b>VAR(Name,Value,Style)</b>	Create a variable of a specified type with a default value	OK	<a href="#">VAR</a>
<b>IF(Var~Var?Func1:Func2)</b>	Evaluate condition and do func1 if true, func2 if false	OK	<a href="#">IF</a>
<b>LOOP(Name,Var){...}</b>	Loop for a specified number of times	OK	<a href="#">LOOP</a>
<b>INT(Name,Buffer,Function)</b>	If interrupt triggered do function	OK	<a href="#">INT</a>
<b>CALC(Result, Var1, Var2, Act)</b>	Quick calculation and text manipulation	OK	<a href="#">CALC</a>
<b>FUNC(Name) {...}</b>	Declare a set of commands	OK	<a href="#">FUNC</a>
<b>STYLE(Name,Type) {...}</b>	Predefine parameters for page entities and variables	OK	
<b>WAIT(Time)</b>	Wait specified milliseconds before next	OK	<a href="#">WAIT</a>
<b>;</b>	Terminate command	OK	<a href="#">SEMI</a>
<b>;;</b>	Refresh current page	OK	<a href="#">DSEMI</a>
<b>[ cmd();cmd();...cmd; ]</b>	Enclose commands as inline function in IF, INT, KEY, RUN	OK	<a href="#">INLINE</a>
<b>POSN(X,Y,Page/Name,Style)</b>	Position cursor or re-position named entity	OK	<a href="#">POSN</a>
<b>PAGE(Name,Style) {...}</b>	Specify contents of page		<a href="#">PAGE</a>
<b>sizeX, sizeY</b>	Specify the size of the page	OK except large size	-
<b>posX, posY</b>	Specify the absolute position on screen	OK	-
<b>back</b>	Specify the background colour of page	OK	-
<b>image</b>	Specify a background image for the page	OK	-
<b>TEXT(Name,Text,Style)</b>	Define text		<a href="#">TEXT</a>
<b>font</b>	The ASCII based + extended fonts	OK	-
<b>size</b>	Size multiplier ie 24x24 to 48x48	OK	-
<b>col</b>	Specify the text color.	OK	-
<b>maxLen</b>	Specify the maximum number per row (Max 512)	OK	-
<b>maxRows</b>	Specify the maximum number of rows (Max 32)	OK	-
<b>curRel</b>	Specify the relative placement of the text	OK	-
<b>rotate</b>	Specify the rotation of the text 0,90,180,270	OK	-
<b>DRAW(Name,X,Y,Style)</b>	Create box, circle, line, pixel, shape		<a href="#">DRAW</a>
<b>type</b>	Specify the type of shape to draw	OK	-
<b>col</b>	Specify the border colour of the shape	OK	-
<b>back</b>	Specify the back colour of the shape	OK	-
<b>width</b>	Specify the border width of the shape	OK	-
<b>sizeX,sizeY</b>	Specify the maximum width and height	OK	-
<b>curRel</b>	Specify the relative placement	OK	-
<b>rotate</b>	Specify the rotation of the shape 0,90,180,270	OK	-
<b>IMG(Name,Source,X,Y,Style)</b>	Image placement and manipulation		<a href="#">IMG</a>
<b>scale</b>	The image can be cropped to centre or scale 2.3.4,8	Plan	-
<b>sizeX, sizeY</b>	Specify the maximum width and height	OK	-
<b>curRel</b>	Specify the relative placement .	OK	-
<b>rotate</b>	Specify the rotation of the image 0,90,180,270	OK	-
<b>KEY</b>	Designation of touch or key matrix		<a href="#">KEY</a>
<b>type</b>	Specify the source of key data - touch or external	OK	-
<b>debounce</b>	Specify the time delay to allow a key	OK	-
<b>delay</b>	Specify the time delay for auto repeat	OK	-
<b>repeat</b>	Specify the time delay for auto repeat	OK	-
<b>action</b>	Specify action point as Down or Up	OK	-
<b>curRel</b>	Specify the relative placement	OK	-
<b>System Setup</b>	set up main display system		<a href="#">SYS</a>
<b>bled = 0 - 100;</b>	set LED backlight 0=OFF, 100=full ON or 1-99	OK - 100 levels on v4+ PCB only	
<b>wdog = 0, 100, 500 or 1000;</b>	set watchdog time to OFF, 100ms, 500ms or 1 second	OK	
<b>encode = s , w, m;</b>	s= single byte ASCII, w=2 byte UNI, m= UTF8	OK	
<b>test=show/hideTouchAreas</b>	show or hide outline view of touch areas on screen.	OK	
<b>calibrate=y, n;</b>	calibrate the touch screen	OK	
<b>rotate= 0 or 180;</b>	set screen orientation with respect to PCB.	OK	
<b>Real Time Clock and Date</b>	Specify real time clock	OK	<a href="#">RTC</a>
<b>active</b>	enable = Y or disable = N		
<b>format</b>	various characters specify the date and time format		
<b>RTCSECS</b>	numeric variable containing seconds (0-59)		
<b>RTCMINS</b>	numeric variable containing minutes (0-59)		
<b>RTCHOURS</b>	numeric variable containing hours (0-23)		
<b>RTCDAYS</b>	numeric variable containing days (1-31)		
<b>RTCMONTHS</b>	numeric variable containing month (1-12)		
<b>RTCYEARS</b>	numeric variable containing year (1900-2099)		
	use LOAD(var,RTC) and LOAD(RTC,varY,";",varM,";",etc		

# iSMART Noritake Itron 5.7" TFT Module

<b>Run Time Counter</b>	Predefined variables which can be set and tested. The runtime counter is continually counting. It is independent of the real time clock.	OK	<a href="#">RUN</a>
	Reset(RUNTIME); resets counter to zero LOAD(CNTSECS,23); set value of seconds		
<b>CNTMILLI</b>	increments every millisecond 0-999		
<b>CNTSECS</b>	increments every second 0-59		
<b>CNTMINS</b>	increments every minute 0-59		
<b>CNTHOURS</b>	increments every hour 0-23		
<b>CNTDAYS</b>	increments every day 0-n		

To update a variable from a port using the equate sign e.g. VOLTS=34.5; will be available from end Jan 2011  
Until then, please use LOAD(VOLTS,34.5);

**System Commands**

Command	Description and Status
<b>FPROG</b> ..... <b>FEND</b>	FPROG and FEND are used to program subsequent commands into internal flash memory. Use the RESET (LIBRARY) command before FPROG if the existing structure is to be replaced, otherwise the commands are appended to the existing structure. <b>Plan.</b>
<b>LIB(Name,Source)</b>	Store image, font, user font or user code file in the library.  <b>Image and Fonts from an SD Card (Onboard Flash)</b> Image and Font files can be BMP and FNT formats. Use iDevTFT to auto convert GIF, JPG, PNG. Since BMP format does not contain transparency information, a colour can be specified after the file name. The rotation and scaling of an image can also be specified as in the IMG command.  <u>Example</u> LIB(myimage,"SDHC/backimg.bmp?back=\\000007"); <b>v0.21.</b> LIB(myimage,"SDHC/backimg.bmp?back=\\000007&rotate=180&scale=75"); <b>v0.21.</b> LIB(asc16x16fnt,"SDHC/asc16B.fnt?start=\\0020"); <b>v0.27</b>  <b>Image and User Font loaded from a Serial Link TBD</b> Where the image or font is sent over a serial interface use the following command structure.  <u>Examples</u> LIB(myimage,"rs2/myimg.bmp?back=\\FFFFFF&rotate=180&scale=75"); LIB(myimage,"rs4/mypic.bmp?back=\\FFFFFF"); LIB(myfont,"spi/fnt?start=\\0000");  <b>User Code TBD</b> User code is submitted in 'C' and compiled by our firmware engineers subject to quotation and agreement. The resultant file is of type .BIN. The user code can then be used with the RUN(Name) command. LIB(myprog,"sdhc/ourprog.bin"); LIB(myprog,"rs2/bin?bytes=36574");  The system does not yet recognize directory structures in the SDHC card. Please put all active files in the root. All file names are 8 characters maximum length.  <b>.BMP is operational v17.</b> <b>User Compiled Code and User Font Array TBD</b>
<b>INC(Source)</b>	Include another menu, style or setup file in the current file. 7 levels of include are possible. This command can be used to reference a file containing styles and commands on the SDHC card so that it's contents are included at that point in the command process. This enables modular design of the menu system.  The system does not recognize directory structures in the SDHC card. Please put all active files in the root. All file names are 8 characters maximum length.  <b>Example:</b> INC("sdhc/submenu.mnu") specifies the file path on the SDcard. INC(File1,File2,File3,...FileN); multiple files are possible <b>Operational</b>
<b>RESET(Name)</b>	Clear the contents of the RunTime Counter, Delete List, Library Files or do a System reset. Reset the System so that it re-boots as at power ON using RESET(SYSTEM) Clear the runtime counter with RESET(RUNTIME); Clear the EEPROM and reload defined variables RESET(EEPROM); Clear the deleted entity list with RESET(DELETED); Clear the library with RESET(LIBRARY); //Allows new program to load. Interface setup unchanged.  <b>Reset 'Deleted' on 16th April</b>
;	Command separator used in menu files and data sent or received via serial interfaces. Example: RUN(HELP); WAIT("1000"); <b>Operational</b>
;;	Refresh the current page. Can be used for refreshing a page after a series of entity updates without knowing which page is showing. LOAD(VOLTS,"34");LOAD(AMPS,"100");; <b>Operational</b>
<b>[ cmd(..); cmd(...);.....cmd(..); ]</b>	The commands which require a function as a parameter ie IF, RUN, INT and KEY can have the function code embedded inside the commands by enclosing the required code in square brackets. This allows you to reduce the number of lines of code for simple functions and where the function is unlikely to be used elsewhere.  <b>Without inline:</b> KEY(keyFlr15,floor15fnc,104,84,TOUCH); //calls function floor15fnc  FUNC(floor15fnc) { LOAD(vReqd,15); TEXT(txtCurFlr,"15"); RUN(fncGo); }  <b>With inline:</b> KEY(keyFlr15, [ LOAD(vReqd,15); TEXT(txtCurFlr,"15"); RUN(fncGo); ],104,84,TOUCH);  <b>Operational</b>

## Page and Group Commands

Command	Description
<b>PAGE(Name,Style) {.....}</b>	<p>Create a Page or Group of entities. Pages contain entities to be shown on the display plus functions that will run as a background task only on that page. Entities are listed so that they are layered from back to front. Create the style and declare the page before using the SHOW(PageName); command.</p> <p><b>Example:</b> In the Aircon example, the main page image has buttons which need a touch area located over each of them. Position the cursor then draw a touch key area. PAGE(MainPage,MainPgStyle) {   POSN( 400, 208 ); KEY( StopKey, StopEvent, 95, 95, TOUCH ); //call function StopEvent   POSN( 76, 252 ); KEY( SaveKey, SaveEvent, 62, 24, TOUCH ); //call function SaveEvent   POSN( +80, +0 ); KEY( CalibKey, CalibEvent, 62, 24, TOUCH ); //call function CalibEvent   POSN( +80, +0 ); KEY( ClockKey, [Show(Clock)];, 62, 24, TOUCH ); //inline code to show clock }</p> <p><b>Page Styles</b> The style defines the page size, position and background. STYLE(stPage,Page) //create a style name and define as type PAGE {   sizeX=480; //specify width of page 1 to 3* LCD width   sizeY=272; //specify height of page 1 to 3* LCD height   posX=0; //specify the absolute X position of page on screen. -4 * LCD width to 4 * LCD width   posY=0; //specify the absolute Y position of page on screen. -4 * LCD height to 4 * LCD height   back=black; //specify background colour of page as hex \000000 to \FFFFFF or colour name   image=pageimg; //specify background image of page as SDHC path or entity name using LIB. }</p> <p><b>Page with screen or smaller.</b></p>
<b>LOAD(Dest,Name,Name,....)</b>	<p>Copy Pages and Groups into a previously defined Page or Group . The background and page attributes for 'Dest' apply to the result so only entities are copied from previous pages. This allows simple templates to be merged to form a complex page.</p> <p>Combine Variables, Buffers and Text and copy the result to a Variable or Buffer. This allows absolute text and variables to be joined together and sent to an interface.</p> <p>Example: LOAD(num,2); //load variable num with value 2 LOAD(EditText,EditText,"D"); //Concatonate contents of EditText with D LOAD(RS2,"DATE=", DTIME, ";", TEMP="",ACTVAL, ";", \0D\0A"); //send concatenated data to RS232 LOAD(NumImg,"Image",num,".bmp"); //Create a name like Image2.bmp LOAD(BasePage,BaseBack,BaseEnglish); //Create page from template pages</p> <p><b>Change Setup Parameters</b> To change setup parameters use the dot operator. Do not change size and watchdog parameters. This operator works for: RS2, RS4, AS1, AS2, DBG, I2C, SPI, PWM, ADC, KEYIO, SYSTEM LOAD( system.bled, 50 ); LOAD( rs2.baud, 9600); LOAD( rs2.baud, baudvar); //use a variable <b>Plan to change style and var parameters in v0.40</b></p> <p><b>Text to Integer/Float</b> LOAD(MyInt,MyText); //The text string is parsed until a non-valid numeric value. LOAD(MyInt,"1","2","3"); //MyInt = 123 If the string does not start with a number or +/- then the result is 0. <b>v0.36</b></p> <p><b>Example Pointers</b> To set/change which entity the entity pointer is pointing to you use '&gt;' instead of ','. LOAD( EntPtr1&gt;"Var1" ); // Set EntPtr1 to point to Var1 LOAD( EntPtr1&gt;"Var1",num,"3" ); // Set EntPtr1 to point to Var123 (very power full not found in C)</p> <p>To put data or an entity name into the entity pointed to by the entity pointer use quotes. LOAD( EntPtr1, "ABC" ); // Load the Entity pointed to by EntPtr1 with "ABC"</p> <p><b>Operational</b></p>
<b>SHOW(Name)</b>	<p>Show a Page on the Display or reveal a hidden Group or Entity This puts the selected page on the top layer of the screen. If the HIDE() command has previously been used for an entity, it will now appear on a page when the page is shown on the display. Show(Page) can also used to refresh a page if entities have changed.</p> <p>Reserved names provide relative navigation when the name of a page may not be known.. Show(PREV_PAGE); Show the page which launched the current page. Show(THIS_PAGE); Refresh the current page Show(Entity1, Entity2, Entity3...); multiple show entities then refresh current page</p> <p><b>Operational</b></p>
<b>HIDE(Name)</b>	<p>Hide a Page, Group or Entity. If the page on which a small sized page, group or entity is placed is showing on the screen and the page refreshed, the named page, group or entity will disappear from view. Touch, external keys are disabled.</p> <p>Hide(Entity1, Entity2, Entity3...); multiple hide entities then refresh current page</p> <p><b>Operational</b></p>
<b>DEL(Name)</b>	<p>Delete a Page, Group, Entity, Variable or Buffer from SDRAM. If visible on the display, it will remain until the page is refreshed. If the name refers to an image, font or file stored in the flash library then this is set for memory to be freed using RESET(DELETED); The command DEL("LIBRARY") is used prior to renewing all the application files.</p> <p>Del(Entity1, Entity2, Entity3...); multiple delete entities</p> <p><b>Delete is operational V17</b> <b>The function RESET( DELETED ) to free memory is not available until Feb2011</b></p>

**Commands for Cursor Position, Text, Draw, Image & Keys - 15****Commands for Cursor Position, Text, Draw, Image and Keys**

Command	Description
<b>POSN(X,Y,Page/Name,Style)</b>	<p>Position Cursor +X or -X or X,Y or X, Y, Page with a defined style. The cursor can be positioned on the display using absolute co-ordinates or moved in relation to it's current position by using +/- offset values. The origin is located at the top left of the screen.</p> <p>Re-position a previously placed entity by specifying the new coo-ordinates and it's name. This can be useful for indicator bars, simple movement animations and moving text.</p> <p><b>Examples:</b>            POSN(+25,+0); moves the cursor 25 pixels to the right.            POSN(236,48); absolute position of x=236, y=48.            POSN(24,56,CalcPage); position cursor on calc page at x=24, y=56.            POSN(VarX,Vary); use variables with absolute values to control position of cursor            POSN(VarX,Vary,VertBar); use variables to move an entity - vertical bar            POSN(TOUCHX,TOUCHY,MyRectCursor); move a cursor to the contact point on the screen.</p> <p><b>Operational</b></p>
<b>TEXT(Name,Text,Style)</b>	<p>Create or update Text. Use Carriage Return and Line Feed for multi line entry <code>\0A\0D</code> The font and colour are defined in the style. If the cursor relative position is 'CC' (Centre Centre) it is easy to locate text in the centre of images like buttons. Text areas can overlap other text areas when for example a 'drop shadow' is required. Text can include embedded hex codes to access Unicode fonts and a cursor.</p> <p><b>Examples:</b>            TEXT(EditBox,"Hello World",st8Red12); //creates Edit Box with user defined style st8Red12            TEXT(EditBox,"Hello People"); //modifies content of EditBox            TEXT(EditBox,TextVar); //modifies content of EditBox with content of variable            TEXT(EditBox,"Hello\w0020World"); // example of unicode embedded character (see fonts page)</p> <p><b>Editable Text and Visible Cursor</b>            A text can contain single byte hex of the form <code>\00</code> to <code>\FF</code>            A text can contain hidden codes for use in password and editable fields.  <code>\01</code> defines the text as a PASSWORD so that only ***** are shown.  <code>\02</code> defines a hidden cursor and <code>\03</code> a hidden cursor with insert ON  <code>\04</code> defines an underline cursor and <code>\05</code> an underline cursor with insert ON  <code>\06</code> defines a block cursor and <code>\07</code> a vertical cursor with insert ON            Always place the cursor before the applicable character.            When a page or text is hidden, the cursor remains at it's current location.            The CALC command can then be used to manipulate the text and cursor in EditBox.</p> <p><b>Example Editable Text:</b>            TEXT(EditBox,"Hello <code>\04World</code>",8ptTextRed); this places an underline cursor at W</p> <p><b>TEXT Styles</b>            Fonts are available using single byte, 2 byte and UTF8 multi-byte coding.            Built in ASCII fonts have the reserved names Ascii8, Ascii16, Ascii32 (case sensitive).            Other library fonts are uploaded using the LIB command and have file type .FNT            These are available for download from the character fonts web page at <a href="http://www.itrontft.com">www.itrontft.com</a>.  <b>Unique Font Overlay</b>            It is possible to overlay one font over another to enable single byte operation with ASCII from 20H to 7FH and Cyrillic, Greek, Hebrew, Bengali, Tamil, Thai or Katakana from 80H to FFH. The LIB command is used to load the extended font at 0080H instead of it's normal UNICODE location. The style for a text can then specify <code>font="MyASCII,MyThai"</code>; causing the Thai to overlap the ASCII from 80H to FFH.</p> <pre>STYLE(Txt32ASC16,TEXT) //assign a name for the style like Txt32ASC16 {   font="ASC16B,16THAI"; //define fonts using built in or preloaded .FNT files via LIB command   size=2; //a 24x24 font is expanded to a 48x48 font. default=1   col=white; //"\000000" to "\FFFFFF" or reserved words from the colour chart.   maxLen=64; //maximum length of text. default =32, maximum=512   maxRows=4; //maximum number of rows=32 where new line code \0D\0A is used.   rotate=90; //rotation relative to screen 0, 90, 180, 270. default=0   curRel=CC; //specify placement relative to cursor. CC Centre Centre, TC Top Centre, } //BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left, // BL Bottom Left, TR Top Right, BR Bottom Right</pre> <p><b>Operational</b></p>
<b>DRAW(Name,X,Y,Style)</b>	<p>Draw or update a Line, Box, Circle or Graph of size X,Y. The entities can be an outline or filled. The colour can be enhanced using alpha blending within the draw style. Graphs of a different colour can be superimposed on top of each other.</p> <p>DRAW accepts VARs, signed/unsigned integers (U8, U16, U32, S8, S16, S32), floats (FLT) and pointers (PTR)</p> <p>DRAW( PTR, VAR INT FLT PTR, VAR INT FLT PTR, Style ); Note PTR refers to the entity being pointed to by PTR and not the PTR itself. Use LOAD( PTR &gt; "Name" ); to set a pointer.</p> <p><b>Example Draw</b>            DRAW(MyCircle, 32, 32, DrawCircle);            DRAW(MyCircle, 64, 64); //modified circle is double diameter.            DRAW(MyBox,VarX,VarY); //modified box using variables. Should not exceed MaxX,maxY.</p> <p>DRAW(MyLine,10,10,lineStyle); //draws line 45 degrees top left to bottom right.            DRAW(MyLine2,10,-10,lineStyle); //draws line 45 degrees bottom left to top right.</p> <p>Graph            DRAW(MyGraph,100,100,GraphStyle); //draws a graph window of 100x100 pixels.            DRAW(MyGraph,20,30); //draws a pixel on the graph at 20,30 relative to the origin.            DRAW(MyGraph,varX,varY); //use variables to plot a pixel on the graph.            RESET(MyGraph); //clears the graph</p> <p>Please refer to the ADC analogue input section for an <a href="#">application example</a>.</p> <p><b>Draw Styles</b>            It is possible to specify transparency values with colours if the colour is entered as a 32-bit hex number the top 8</p>



# iSMART Noritake Itron 5.7" TFT Module

	<p>bits specify the alpha blending level.  <code>col = \aarrggbb; back = \aarrggbb; where aa = alpha level.</code>          For example, <code>col = \80FFFF00;</code> gives 50% transparent yellow.</p> <pre>STYLE(stCircleRed,DRAW) {   type=B; //Specify the type of shape to draw. <b>type</b> = B or Box , C or Circle, L or Line, G or Graph   col=red; //Specify the border colour of the shape. Use hex, colour name + alpha   width=1; //Specify the border width of the shape default = 1   back=\00FF66; //Specify the fill colour of the shape. Use hex, colour name + alpha   maxX=160; // Declare the maximum width allowing for rotation   maxY=40; // Declare the maximum height allowing for rotation   rotate=0; // Specify the rotation of the shape with respect to the screen. 0,90,180,270   curRel=CC; //specify placement relative to cursor. CC Centre Centre , TC Top Centre, } //BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left, // BL Bottom Left, TR Top Right, BR Bottom Right</pre> <p><b>Operational</b></p>
<p><b>IMG(Name,Source,Style)</b></p>	<p>Draw or update an Image of size X,Y. Source has several techniques.          If an image is pre-stored in the library, it's entity name is used for Source.          If it is to be directly loaded from the SDHC card the path is the Source.          Scaling and rotation can also be specified in the LIB command.          The system does not recognize directory structures in the SDHC card.          Please put all active files in the root. All file names are 8 characters maximum length.          Old version IMG(name,source,x,y,style); supported although x,y not used.</p> <p><b>Example:</b>  <code>IMG(MyPic,TopBtnMyImage); //previously stored as TopBtn using LIB command</code>  <code>IMG(MyPic,"sdhc/TopBtn.bmp",90,60,MyImage); //stored on SDHC card</code></p> <p><b>Image Styles</b>          The image may be larger than the size specified so it is necessary to define how it will be scaled.  <code>STYLE(MyImage,Image)</code></p> <pre>{   scale=100; // The image is scaled down or up by a percentage. //Supports 5% steps below 100 and 100% steps above 100.   maxX=160; // Declare the maximum width allowing for rotation   maxY=40; // Declare the maximum height allowing for rotation   rotate=0; // Specify the rotation of the shape with respect to the screen. 0,90,180,270   curRel=CC; // specify placement relative to cursor. CC Centre Centre , TC Top Centre, } // BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left, // BL Bottom Left, TR Top Right, BR Bottom Right</pre> <p><b>Operational except scale C.</b></p>
<p><b>KEY(Name,Function,X,Y,Style)</b></p>	<p>Create a Touch Area of size X,Y or define a Key on the external keyboard.</p> <p>The touch area can have a One Touch function by using the built in style TOUCH or TOUCHR (repeat)          An external key can use the default style KEYIO.          Both these built in styles process when the key is depressed.          For processing at press and release, create 2 keys at the same location with different styles, one with  <code>action=DOWN;</code> and the other with <code>action=UP;</code></p> <p>When specifying an external key action, the values for X and Y indicate the contact points on the key board matrix          where K0 is <code>\00</code> through to K23 which is <code>\17</code>.</p> <p>This method allows dual key press capability as in SHIFT key operation.          Key scan uses ports K0-K23 which can be configured as shown in the I/O section.          Switches connected to 0V should use the I/O interrupt command <code>INT(...);</code></p> <p>The last touch co-ordinates are stored in predefined variables TOUCHX and TOUCHY</p> <p>The touch screen can be calibrated using the command <code>setup( system ) { calibrate=y; }</code>          The position of touch keys can be temporarily viewed as a grey area using  <code>setup( system ) { test=showTouchAreas; }</code> and hidden again using <code>test=hideTouchAreas.</code>          The built in style TOUCHR provides auto repeat after 1sec with 200ms repetition.          See the SYSTEM command for global touch screen debounce, sampling and accuracy parameters.</p> <p><b>Examples KEY</b>  <code>Key(TopKey,TopFnc,90,50,MyTouch);</code> a touch area 90x50 pixels. Create your own style MyTouch  <code>Key(ExtKey,ExFunc,\07,\10,KEYIO);</code> This external key operates when K7 and K16 connect.  <code>Key(TKey,[Hide(SPage);Show(TPage);],50,50,TOUCH);</code> Inline commands instead of function</p> <p><b>Plan:</b> <code>Key(ExtKey,ExFunc,K07,K16,PushKey);</code> This external key operates when K7 and K16 connect.</p> <p><b>KEY Styles</b>          Specify the source of key data. Touch debounce and sampling is setup globally in SYSTEM          If you require a dual action, specify 2 keys at the same location, one with action D and one with U.</p> <pre>STYLE(myTouch,key) {   type=touch; //specify 'touch' screen or external 'keyio'   debounce=250; //Specify the time delay to allow external key press to stabilise in milliseconds.   delay=1000; //Specify the time delay before key auto repeat occurs in milliseconds. 0=off.   repeat=500; //Specify the repeat period if the key is held down in milliseconds   action = D; //Specify D or Down and U or Up. Specify the up or down action point for the key.   curRel=CC; //specify touch key placement relative to cursor. CC Centre Centre , TC Top Centre, } //BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left, // BL Bottom Left, TR Top Right, BR Bottom Right</pre> <p><b>Operational.</b></p>

Function Commands

Command	Description
<b>RUN(Name)</b>	Run previously defined user code or functions. User code is supplied in C and compiled by our firmware department subject to order. Functions can be run as macros for compact menu design. RUN(Func1); or RUN(Func1,Func23,Func3...FuncN); or a pointer to a function RUN(func-ptr); <b>Operational except User code TBD.</b>
<b>WAIT(Time)</b>	Wait for a period of milliseconds before processing menu commands. Interrupts and key presses will still occur and can be processed. Wait timer accuracy increased, now running off system tick timer (max error 200ns). <b>Wait is operational.</b>
<b>FUNC(Name) {...}</b>	Create a function called by commands which returns to the next command on completion. Functions can call other functions and themselves. No storing or passing of variables occurs as these are all global even if created in a function. Max 12 nested loops or functions. <b>Operational</b>
<b>VAR(Name,Value,Style)</b> <b>+ pointer usage</b> <b>+ non volatile parameter storage</b>	Create a variable having a certain style and a default value. A variable contains text or numbers which can be amended and be referred to as a single name in an equation or to show information on the display. Variable names must start with a letter or _. Variables can be pointers to other variables and entities and use the '>' operator. Non volatile parameter storage is also handled by VAR which initially loads the default value, then at subsequent power ON reloads the last stored value which was saved using LOAD(varname,newval); <u>Example Numbers</u> VAR(lowval,32.4,FLT1); define lowval as a single decimal float and default value 32.4 VAR(lowval,22.4,FLT1E); define lowval as a single decimal float and default value 22.7 or load EEPROM value if already exists. Use RESET(EEPROM); to clear and reload only current values. <u>Example Pointers</u> Create a pointer which is defaulted to null using the '>' symbol. VAR(EntPtr1>"",PTR);  To set/change which entity the entity pointer is pointing to you use '>' instead of ','. LOAD( EntPtr1>"Var1"); // Set EntPtr1 to point to Var1  To put data into the entity pointed to by the entity pointer, enclose data / source entity in quotes. LOAD( EntPtr1, "ABC" ); // Load the Entity pointed to by EntPtr1 with ABC  The following commands now support entity pointers where (   means 'or this') > LOAD(name   ptr   "ptr",   > num   "txt"   var   ptr,...); > CALC(var   ptr, var   ptr, num   var   ptr,"op"); > TEXT(name   ptr, "txt"   var   ptr,...); > IF(var   ptr op num   "txt"   var   ptr ? func   func_ptr : func   func_ptr); > KEY(name, func   func_ptr,...); > INT(name, buf, func   func_ptr,...); > SHOW(name   ptr,...); > HIDE(name   ptr,...); > RUN(name   func_ptr,...); > IMG(name   img_ptr, lib   img_ptr,...);  <b>VAR Data Styles</b> Specify your own style for integer, float, pointer or text or use a built in style name  STYLE(stVar, Data) { type = U8; // U8, U16, U32 - unsigned 8, 16 and 32 bit integer // S8, S16, S32 - signed 8, 16, 32 bit integer // TEXT for text strings // FLOAT for higher resolution calculation // POINTER for use with images length=64; // For text, specify the length from 1 to 8192, default =32 decimal=3; // Specify the number of decimal places when type is float. Range 0 to 7, default=2 format="dd mm YY"; //Specify RTC format. see RTC page for format character types location=SDRAM; //Specify the data location as SDRAM (default) or EEPROM }  <u>Built In Styles (Add E for EEPROM types Example FLT4E)</u> The following pre defined 'built in' style names are available <b>U8/U8E</b> - type = U8, <b>U16/U16E</b> - type = U16, <b>U32/U32E</b> - type = U32 <b>S8/S8E</b> - type = S8, <b>S16/S16E</b> - type = S16, <b>S32/S32E</b> - type = S32 <b>PTR/PTR</b> - type = pointer, <b>TXT/TXTE</b> - type = TEXT, length=32 <b>FLT1/FLT1E</b> - type = float, decimal = 1, <b>FLT2/FLT2E</b> - type = float, decimal = 2 <b>FLT3/FLT3E</b> - type = float, decimal = 3, <b>FLT4/FLT4E</b> - type = float, decimal = 4  <b>Operational</b>
<b>IF(Var~Var?Function1:Function2)</b>	Compare variables, buffers or text for value or length. If true, do function1, if false do function2 (optional). The ~ operator types can compare text length with another text or a numeric length.  The operators allowed for numeric values are: =, == equal to <>, != not equal to < less than > greater than <= less than or equal to >= greater than or equal to + sum not equal to zero - difference not equal to zero * multiplication not equal to zero / division not equal to zero % modulus not equal to zero & logical AND   logical OR ^ logical exclusive-OR =- equal to the negative of && Boolean AND    Boolean OR  The operators allowed for text strings are: =, == equal to > greater than < less than >= greater than or equal to <= less than or equal to <>, != not equal ~= same text length ~< text length shorter than ~> text length longer than ~! not same text length

# iSMART Noritake Itron 5.7" TFT Module

	<p><b>Examples:</b>  IF(K0="L"?HELPFNC); //single condition  IF(HIGHVAL &lt; ACTVAL ? HIGHFUNC : LOWFUNC);  IF(STRVAR~&gt;0? SHOWFUNC); //if STRVAR length &gt; 0 show data  IF(STARVAL &gt;= -STARTMP?SHOWSTAR);  IF(STARVAL &gt; 0? [ LOAD(vReqd,15); TEXT(txtCurFlr,"15"); RUN(fncGo); ] ); //uses in line code [...]</p> <p><b>Operational v18</b></p>
<p><b>LOOP(Name,Var1){.....}</b></p>	<p>Repeats the specified actions a number of times in a PAGE then continue. Max 12 nested loops or functions. The value for Var1 can be a number from 1-65000 or the text FOREVER. You can terminate a LOOP using the command DEL(Name);</p> <p><b>Examples:</b>  LOOP(MyLoop,12){Show(Page1);wait(100);show(page2);wait(100);} //repeat 12 times  LOOP(MyLoop,FOREVER) {Show(Page1);wait(100);show(page2);wait(100);}</p> <p><b>Operational v17 for use in PAGE. Use in FUNC expected 4th Feb.</b></p>
<p><b>INT(Name,Buffer,Function)</b></p>	<p>If an interrupt occurs for the specified buffer, do function.  An interrupt will occur when a buffer's style parameters allow activity within the buffer and the appropriate type of interrupt is set.  Serial interfaces can trigger on a byte received, a byte transmitted and a semi-colon (command separator) received. I/O can trigger on input change.  Use HIDE(Name); to disable an interrupt.</p> <p>This is currently set to interrupt on each character received for the 'Buffer':  &gt; RS2RXC = RS232 Receive Character  &gt; RS4RXC = RS485 Receive Character  &gt; AS1RXC = Async1 Receive Character  &gt; AS2RXC = Async2 Receive Character  &gt; DBGRXC = Debug Receive Character  &gt; I2CRXC = I2C Receive Character</p> <p><b>NOTE: The Buffer must be read to clear the interrupt otherwise the Function will keep getting called!</b></p> <p><b>Example:</b>  PAGE( PageName, PageStyle)  {  INT( SerRxInt, RS2RXC, SerRxEvent );  }  FUNC( SerRxEvent )  {  LOAD( Var, RS2 ); // Must read RS2 to clear interrupt  LOAD( RS4, Var); //send out of RS485 interface.  TEXT ( RecvTxt, Var); //show received ASCII data on screen  // and refresh  }</p> <p><b>Operational except for counters</b></p>
<p><b>CALC(Result,VarA,VarB,Method)</b></p>	<p><b>Numeric Handling</b>  This provides a fast simple calculation placed in the Result variable according to the type of method using +, -, /, *, %(modulus) or logical functions   (OR) &amp; (AND) ^ (EXOR) for non float. Trig functions are planned.</p> <p>Example: CALC(NumUp,NumUp,1,"+"); increments NumUp  CALC(FltNum,1.0,FltVal,""); first parameter defines type for 2nd and 3rd parameter.  CALC(Result,Request,8,"&amp;"); the Result will equal 0 or 8 if bit3 is set in Request.  CALC(SumPtr,PtrA,PtrB,"/"); use pointers for the calculation</p> <p><b>Text and Cursor Handling</b>  Calc can be used for text and cursor manipulation where editable text is to be placed on the screen as in a calculator or editable text field. Various methods allow cursor movement and type, text insertion and deletion, find or delete text, cursor position and length.  VarA contains the existing text and VarB the modifier text, cursor position or a text length.  Example: CALC(EditBox,EditBox, "A", "INS"); Inserts the letter 'A' into the text at the cursor position</p> <p><b>Cursor and Text Types</b>  \\01 defines the text as a PASSWORD so that only ***** are shown until another \\01 or end;  \\02 defines a hidden cursor with over write and \\03 a hidden cursor with insert ON  \\04 defines an underline cursor with over write and \\05 an underline cursor with insert ON  \\06 defines a block cursor with over write and \\07 a ertical cursor with insert ON</p> <p><b>Method Types - The first character in a string is 0.</b>  INS Add text in VarB at cursor position according to cursor type and move cursor (Overwrite/Insert)  DEL Delete text of length VarB at cursor position and shift remaining text left  If VarB is negative then text is deleted before the cursor as in Back Space  TRIM Remove characters from the beginning and end of string specified in a list VarB  LTRIM Remove charaters from the start of string as specified in VarB  RTRIM Remove charaters from the end of string as specified in VarB  POS Move cursor to absolute position in text as specified in VarB 0-n  REL Move cursor relative to existing position specified in VarB -n to +n  FIND Result gives the start position of case sensitive text VarB in VarA  IFIND Result gives the start position of case insensitive text VarB in VarA  REM Any case sensitive occurrence of the text VarB in VarA is removed and the text shifted left.  IREM Any case insensitive occurrence of the text VarB in VarA is removed and the text shifted left.  SPLIT Scans the string for a character and puts first part in result with remainder in VarA  CUR The cursor or text type is changed at the current position to type VarB (\\01 to \\07)  LEN Result contains the current length of the text in characters plus VarB.  PIXX Result contains the current length of the named text entity in pixels plus VarB.  PIXY Result contains the current height of the named text entity in pixels plus VarB.  LOC Result contains the position of the cursor in the text plus offset in VarB (-n to +n)  TYPE Result contains the type of text and cursor used - \\01 to \\07 or \\00 if none present.  AFT Result contains VarB characters after cursor position in string VarA. If no cursor, use first  Example CALC(result,"abc\\02defghij",4,"AFT"); result="defg"  BEF Result contains VarB characters before cursor position in string VarA. If no cursor, use end  Example CALC(result,"abc\\02defghij",2,"BEF"); result="bc"  UPPER Convert string VarA to upper case  LOWER Convert string VarA to lower case</p> <p><b>"POS" - Move Cursor to Absolute Position</b>  CALC( dst, src, pos, "POS" );  Moves cursor in text 'src' to absolute position 'pos' and stores result text in 'dst'.  If 'pos' is less than zero, then cursor is put before first character ('pos'=0). If 'pos' is greater than</p>

the length of 'src' then the cursor is placed after the last character.

'dst' and 'src' can be the same text variable.

'src' is unmodified unless same text variable as 'dst'.

Supported data types:

'dst' => text variable | pointer to text variable

'src' => text variable | pointer to text variable | "string"

'pos' => integer variable | pointer to integer variable | integer

#### "REL" - Move Cursor to Relative Position

CALC( dst, src, mov, "REL" );

Moves cursor in text 'src' by displacement specified in 'mov' and stores result text in 'dst'.

Positive values of 'mov' move the cursor to the right and negative values of 'mov' move the cursor to the left.

If the move results in a cursor position of less than zero, then the cursor is put before first character. If the move

results in a cursor position greater than the length of 'src' then the cursor is placed after the last character.

'dst' and 'src' can be the same text variable.

'src' is unmodified unless same text variable as 'dst'.

Supported data types:

'dst' => text variable | pointer to text variable

'src' => text variable | pointer to text variable | "string"

'mov' => integer variable | pointer to integer variable | integer

#### "INS" - Insert / Overwrite Text at Cursor

CALC( dst, src1, src2, "INS" );

Puts text from 'src2' into 'src1' at the cursor and stores the result text in 'dst'.

The text will either be overwritten or inserted depending on the cursor type in 'src1'.

If no cursor is present then the text is appended to the end of 'src1'.

'dst' and 'src1' can be the same text variable.

'src1' and 'src2' are unmodified unless same text variable as 'dst'

Supported data types:

'dst' => text variable | pointer to text variable

'src1' => text variable | pointer to text variable | "string"

'src2' => text variable | pointer to text variable | "string"

#### "DEL" - Delete Text at Cursor

CALC( dst, src, num, "DEL" );

Deletes 'num' characters from text 'src' at the cursor and stores the result text in 'dst'.

If 'num' is positive, then 'num' characters will be deleted after cursor. If 'num' is negative, then '-num'

characters will be deleted before cursor (backspace).

If no cursor is present and 'num' is negative, then '-num' characters will be deleted from the end of the text in

'src'. If no cursor is present and 'num' is positive, then 'num' characters will be deleted from the start of the

text in 'src'.

'dst' and 'src' can be the same text variable.

'src' is unmodified unless same text variable as 'dst'.

Supported data types:

'dst' => text variable | pointer to text variable

'src' => text variable | pointer to text variable | "string"

'num' => integer variable | pointer to integer variable | integer

#### "TRIM" - Trim Characters from Start and End of Text String

CALC( dst, src, list, "TRIM" );

Removes all text characters found in 'list' from the start and end of text in 'src' and stores the result text in

'dst'.

If 'list' is "" (empty string) then spaces (20hex), tabs (09hex), line feeds (0Ahex), and carriage returns

(0Dhex) are removed.

'dst' and 'src' can be the same text variable.

'src' is unmodified unless same text variable as 'dst'.

Supported data types:

'dst' => text variable | pointer to text variable

'src' => text variable | pointer to text variable | "string"

'list' => text variable | pointer to text variable | "string"

#### "LTRIM" - Trim Characters from Start of Text String

CALC( dst, src, list, "LTRIM" );

Removes all text characters found in 'list' from the start of text in 'src' and stores the result text in 'dst'.

If 'list' is "" (empty string) then spaces (20hex), tabs (09hex), line feeds (0Ahex), and carriage returns

(0Dhex) are removed.

'dst' and 'src' can be the same text variable.

'src' is unmodified unless same text variable as 'dst'.

Supported data types:

'dst' => text variable | pointer to text variable

'src' => text variable | pointer to text variable | "string"

'list' => text variable | pointer to text variable | "string"

#### "RTRIM" - Trim Characters from End of Text String

CALC( dst, src, list, "RTRIM" );

Removes all text characters found in 'list' from the end of text in 'src' and stores the result text in 'dst'.

If 'list' is "" (empty string) then spaces (20hex), tabs (09hex), line feeds (0Ahex), and carriage returns

(0Dhex) are removed.

'dst' and 'src' can be the same text variable.

'src' is unmodified unless same text variable as 'dst'.

Supported data types:

'dst' => text variable | pointer to text variable

'src' => text variable | pointer to text variable | "string"

'list' => text variable | pointer to text variable | "string"

#### "UPPER" - Convert Text to Uppercase

CALC( dst, src, 0, "UPPER" );

Converts the characters 'a'-'z' to uppercase 'A'-'Z' in text 'src' and stores result text in 'dst'.

'dst' and 'src' can be the same text variable.

'src' is unmodified unless same text variable as 'dst'.

Supported data types:-

'dst' => text variable | pointer to text variable

'src' => text variable | pointer to text variable | "string"

#### "LOWER" - Convert Text to Lowercase

`CALC( dst, src, 0, "LOWER" );`  
 Converts the characters 'A'-'Z' to lowercase 'a'-'z' in text 'src' and stores result text in 'dst'.  
 'dst' and 'src' can be the same text variable.  
 'src' is unmodified unless same text variable as 'dst'.  
 Supported data types:  
 'dst' => text variable | pointer to text variable  
 'src' => text variable | pointer to text variable | "string"

*"BEF" - Get Characters from Before Cursor*  
`CALC( dst, src, num, "BEF" );`  
 'num' characters are copied from before the cursor in text 'src' and stored in text 'dst'.  
 If no cursor is present then 'num' characters are copied from the end of 'src'.  
 If 'num' is larger than the number of characters available in 'src' then only the available characters are copied.  
 If 'num' is negative, then the function performs as "AFT".  
 'dst' and 'src' can be the same text variable.  
 'src' is unmodified unless same text variable as 'dst'.  
 Supported data types:  
 'dst' => text variable | pointer to text variable  
 'src' => text variable | pointer to text variable | "string"  
 'num' => integer variable | pointer to integer variable | integer

*"AFT" - Get Characters from After Cursor*  
`CALC( dst, src, num, "AFT" );`  
 'num' characters are copied from after the cursor in text 'src' and stored in text 'dst'.  
 If no cursor is present then 'num' characters are copied from the start of 'src'.  
 If 'num' is larger than the number of characters available in 'src' then only the available characters are copied.  
 If 'num' is negative, then the function performs as "BEF".  
 'dst' and 'src' can be the same text variable.  
 'src' is unmodified unless same text variable as 'dst'.  
 Supported data types:  
 'dst' => text variable | pointer to text variable  
 'src' => text variable | pointer to text variable | "string"  
 'num' => integer variable | pointer to integer variable | integer

*"CUR" - Change Cursor Type*  
`CALC( dst, src, type, "CUR" );`  
 The cursor in text 'src' is changed to type 'type' and the result is stored in text 'dst'.  
 If no cursor is present, then the new cursor is appended to the end.  
 'dst' and 'src' can be the same text variable.  
 'src' is unmodified unless same text variable as 'dst'.  
 If 'type' is a string then the first character is taken as the cursor type.  
 Supported data types:  
 'dst' => text variable | pointer to text variable  
 'src' => text variable | pointer to text variable | "string"  
 'type' => integer variable | pointer to integer variable | integer | text variable | pointer to text variable | "string"

*"LEN" - Get Text Length*  
`CALC( len, src, num, "LEN" );`  
 The length of text 'src' plus 'num' is stored in variable 'len'.  
 Cursor characters are not included in the length.  
 'src' is unmodified.  
 Supported data types:  
 'len' => integer variable | pointer to integer variable | float variable | pointer to a float variable  
 'src' => text variable | pointer to text variable | "string"  
 'num' => integer variable | pointer to integer variable | integer | float variable | pointer to a float variable | float

*"LOC" - Get Cursor Location*  
`CALC( loc, src, num, "LOC" );`  
 The location of the cursor in text 'src' plus 'num' is stored in variable 'loc'.  
 If no cursor is present then a value of 0 is used.  
 'src' is unmodified.  
 Supported data types:  
 'loc' => integer variable | pointer to integer variable | float variable | pointer to a float variable  
 'src' => text variable | pointer to text variable | "string"  
 'num' => integer variable | pointer to integer variable | integer | float variable | pointer to a float variable | float

*"TYPE" - Get Cursor Type*  
`CALC( type, src, 0, "TYPE" );`  
 The cursor type in text 'src' is stored in variable 'type'.  
 If no cursor is present then a value of 0 is used.  
 'src' is unmodified.  
 Supported data types:  
 'type' => integer variable | pointer to integer variable | float variable | pointer to a float variable  
 'src' => text variable | pointer to text variable | "string"

*"FIND" - Find Location of Text1 in Text2*  
`CALC( loc, src1, src2, "FIND" );`  
 The first location of the match of text 'src2' (needle) in text 'src1' (haystack) is returned in 'loc'.  
 If no matches are found then -1 is returned in 'loc'.  
 Cursor characters are not included in the calculation.  
 'src1' and 'src2' are unmodified.  
 Supported data types:  
 'loc' => integer variable | pointer to integer variable | float variable | pointer to a float variable  
 'src1' => text variable | pointer to text variable | "string"  
 'src2' => text variable | pointer to text variable | "string"

*"IFIND" - Find Location of Case Insensitive Text1 in Text2*  
`CALC( loc, src1, src2, "IFIND" );`  
 The first location of the case insensitive match of text 'src2' (needle) in text 'src1' (haystack) is returned in 'loc'.  
 If no case insensitive matches are found then -1 is returned in 'loc'.  
 Cursor characters are not included in the calculation.

## iSMART Noritake Itron 5.7" TFT Module

'src1' and 'src2' are unmodified.  
Supported data types:  
'loc' => integer variable | pointer to integer variable | float variable | pointer to a float variable  
'src1' => text variable | pointer to text variable | "string"  
'src2' => text variable | pointer to text variable | "string"

"REM" - Remove Every Text1 in Text2  
CALC( dst, src1, src2, "REM" );  
Remove every occurrence of text 'src2' (needle) from text 'src1' (haystack) and store the result text in 'dst'.  
'dst' and 'src1' can be the same text variable.  
'src1' and 'src2' are unmodified unless same text variable as 'dst'.  
Supported data types:  
'dst' => text variable | pointer to text variable  
'src1' => text variable | pointer to text variable | "string"  
'src2' => text variable | pointer to text variable | "string"

"IREM" - Remove Every Case Insensitive Text1 in Text2  
CALC( dst, src1, src2, "IREM" );  
Remove every case insensitive occurrence of text 'src2' (needle) from text 'src1' (haystack) and store the result text in 'dst'.  
'dst' and 'src1' can be the same text variable.  
'src1' and 'src2' are unmodified unless same text variable as 'dst'.  
Supported data types:  
'dst' => text variable | pointer to text variable  
'src1' => text variable | pointer to text variable | "string"  
'src2' => text variable | pointer to text variable | "string"

"SPLIT" - Split Text at Character  
CALC( dst, src, char, "SPLIT" );  
CALC( num, src, char, "SPLIT" );  
Split the text 'src' at the character 'char' storing the text after 'char' back into 'src' and storing the text before 'char' into 'dst' or converting to number 'num'.  
If no 'char' is present then the whole of 'src' is processed.  
If 'char' is a string then the first character is taken as the split character.  
'src' is modified during this operation.  
Supported data types:  
'dst' => text variable | pointer to text variable  
'num' => integer variable | pointer to integer variable | float variable | pointer to a float variable  
'src' => text variable | pointer to text variable | "string"  
'char' => integer variable | pointer to integer variable | integer | text variable | pointer to text variable | "string"

"PIXX" - Get Width of Entity  
CALC( size, ent, num, "PIXX" );  
The display width in pixels of entity 'ent' plus 'num' is stored in 'size'.  
Note, variables do not have a size and return 0. Text, image, draw, touch keys, and pages do have sizes.  
Supported data types:  
'size' => integer variable | pointer to integer variable | float variable | pointer to a float variable  
'ent' => entity name | pointer to entity name  
'num' => integer variable | pointer to integer variable | float variable | pointer to a float variable

"PIXY" - Get Height of Entity  
CALC( size, ent, num, "PIXY" );  
The display height in pixels of entity 'ent' plus 'num' is stored in 'size'.  
Note, variables do not have a size and return 0. Text, image, draw, touch keys, and pages do have sizes.  
Supported data types:  
'size' => integer variable | pointer to integer variable | float variable | pointer to a float variable  
'ent' => entity name | pointer to entity name  
'num' => integer variable | pointer to integer variable | float variable | pointer to a float variable

**Operational v37 except Trig functions Sin, Cos, Tan, CoTan TBA.**

## iSMART TFT Reserved Words

Words	Description
;	Terminate command
::	Refresh current page
ac97	audio buffer. adaptor connects to CN4
action	Specify action point as Down or Up. Used in Key settings
active	I2C -- used Master (M), Slave (S) or None (N) Key I/O -- high is active "\000000" > "\FFFFFF" PWM / ADC -- None (N), 1 (1), 2 (2), both (12) RTC -- enable (Y) or disable (N)
adc1	analogue to digital converter 1 processes at 1000 samples per second
adc2	analogue to digital converter 2 processes at 1000 samples per second
addr	address pair where =nn write, =nn+1 read. Used with i2c interfaces
as1	async1 interface
as2	async2 interface
AS1RXC	Async1 Receive Character
AS2RXC	Async2 Receive Character
avg1	number of samples taken and averaged for ADC1 (1-16)
avg2	number of samples taken and averaged for ADC2 (1-16)
back	Specify the back colour of the object
baud	= 110 to 115200. Used for asynchronous interfaces
bled	LED Backlight 0=OFF 100=FULL or use 1-99
buzz	buzzer output
CALC	Quick calculation and text manipulation eg. CALC(Result, Var1, Var2, Act)
calib1	set user function to use for calibrate/scale ADC1
calib2	set user function to use for calibrate/scale ADC2
calibrate	used in setup(system) to calibrate touch screen
CAN	CANBUS adaptor - 1Mhz - adaptor connects to CN3
CNTDAYS	increments every day (0-n) Used with Runtime counter
CNTHOURS	increments every hour (0-23) Used with Runtime counter
CNTMILLI	increments every millisecond (0-999) Used with Runtime counter
CNTMINS	increments every minute (0-59) Used with Runtime counter
CNTSECS	increments every second (0-59) Used with Runtime counter
CNTK00-CNTK23	I/O counters which can be set up using interrupt and trig parameters
col	Specify the text or border color
curRel	Specify the relative placement of an object
cycle1	cycle1 value in microseconds
cycle2	cycle2 value in microseconds
data	= 5, 6, 7, 8 Used for asynchronous interfaces
dbg	debugger interface
DBGRCX	Debug Receive Character
debounce	Specify the time delay to allow a key
DEL	Delete a page, entity eg DEL(Name)
delay	Specify the time delay for auto repeat
delay	delay in microseconds between pwm1 and pwm2
DELETED	list of deleted entities
DRAW	Create box, circle, line, pixel, shape eg DRAW(Name,X,Y,Style)
duty1	value as a percentage of High period
duty2	value as a percentage of High period
edge	uses Rising(R) or Falling(F) clock edge
EEPROM	internal EEPROM -- parameter storage using extended variables VarE
encode	single byte of ASCII (s), 2 byte UNI (w), UTF8 (m) Used in system settings
end	byte returned when no data left in buffer. Used with spi and i2c interfaces
FLOAT	High resolution calculation data type
flow	flow control - none (N), hardware (H), software (S) XON XOFF Used with asynchronous interfaces
font	The ASCII based + extended fonts
format	various characters specify the date and time format Used the real time clock and date settings
FPROG.....FEND	Store SDHC menu and image files in onboard flash
FUNC	Declare a set of commands eg FUNC(Name) {...}
HIDE	Hide a page, entity eg HIDE(Name)
i2c	i2c interface
I2CRXC	I2C Receive Character
IF	Evaluate condition and do func1 if true, func2 if false eg IF(Var~Var?Func1:Func2)
image	Specify a background image for the page
IMG	Image placement and manipulation eg IMG(Name,Source,Style)
INC	Include the contents of another menu, style or setup file eg INC(FileName)
inp	high is input, low output "\000000" > "\FFFFFF" Used with Key I/O interfaces
INT	If interrupt triggered do function eg INT(Name,Buffer, Function)
KEY	Designation of touch or key matrix
keyb	high is scanned keyboard connection
keyio	K23 is the highest order bit and K0 the lowest
LIB	Load picture or font into library eg LIB(Name,Source)
LIBARY	list of all items stored in the library
LOAD	Multi function copy page, variable N2--N.. to Name. eg LOAD(Name,N2,N3,N..)
LOOP	Loop for a specified number of times eg LOOP(Name,Var){...}
maxLen	Specify the maximum number per row (Max 512)
maxRows	Specify the maximum number of rows (Max 32)
NAND	NAND Flash supports a Proprietary structure
PAGE	Specify contents of page eg PAGE(Name,Style) {...}
parity	= Odd, Even, None, Mark, Space Used with asynchronous interfaces
POINTER	Images data type
poll1	poll1 is High (H) or Low (L) on first phase
poll2	poll2 is High (H) or Low (L) on first phase

## iSMART Noritake Itron 5.7" TFT Module

POSN	Position cursor or re-position named entity eg POSN(X,Y,Page/Name,Style)
posx	Specify the absolute x position on the screen
posY	Specify the absolute y position on screen
proc	process on receive string terminator = ";" or \\0D or other
procDel	delete(Y) or keep(N) termination character.
PTR	entity pointer
pwm	pwm1 , pwm2 - 160Hz to 1MHz
repeat	Specify the time delay for auto repeat
RESET	Clears -- eeprom variable, delete list, library or reset system
rotate	Specify the rotation of the text, shape, image or screen 0,90,180,270
rs2	rs232 interface
rs4	rs485 interface
RS2RXC	RS232 Receive Character
RS4RXC	RS485 Receive Character
rsync	rsync interface
RTC	Real time clock and date
RTCDAYS	numeric variable containing days (1-31)
RTCHOURS	numeric variable containing hours (0-23)
RTCMINS	numeric variable containing minutes (0-59)
RTCMONTHS	numeric variable containing month (1-12)
RTCSECS	numeric variable containing seconds (0-59)
RTCYEARS	numeric variable containing year (1900-2099)
RUN	Run a function or user code eg RUN(Func)
RUNTIME	Runtime Counter The runtime counter is continually counting. It is independent of the real time clock
rxb	set size of receive buffer in bytes. Used with asynchronous, spi and i2c interfaces
rxf	use none (N) or hardware (H) MB. Used with spi interfaces
rxl	set receive buffer interface active ( Y or C or N ) Used with asynchronous, spi and i2c interfaces
rxo	set receive data order ( M or L ) Used with spi interfaces
rxs	use select input \RSS. ( Y or N ) Used with spi interfaces
S16	signed 16 bit integer data type
S32	signed 32 bit integer data type
S8	signed 8 bit integer data type
scale	The image can be cropped to centre or fit
sdhc	SD Card (1G or 4G+ ) FAT32 format - 8 character file names, no directory. Not 2G
set	quick set up combination Used with asynchronous, spi and i2c interfaces
SHOW	Show a page, entity eg SHOW(Name)
size	Size multiplier ie 24x24 to 48x48
sizeX	Specify the maximum width
sizeY	Specify the maximum height
speed	set transmit speed in master mode Used with spi interfaces
spi	spi interface
stop	equals num ( 1, 15, 2 15 is 1.5 bits ) Used with asynchronous interfaces
STYLE	Predefine parameters for page entities and variables eg STYLE(Name,Type) {...}
SYSTEM	Overall settings of the TFT
test	show (showTouchAreas) or hide (hideTouchAreas) outline of touch areas on screen.
TEXT	Define text eg TEXT(Name,Text,Style)
TOUCH	A preset style for TOUCH Key
TOUCHX	contains the last touch Y co-ordinate
TOUCHY	contains the last touch X co-ordinate
trig	high is trigger interrupt
tsync	tsync interface
txb	set size of transmit buffer in bytes. Used with asynchronous, spi and i2c interfaces
txf	none (N) or hardware (H) HB in Master mode. Used with spi interfaces
txl	set transmit buffer interface ( Y or E or N ). Used with AS1/AS2, spi and i2c interfaces
txo	set transmit data order ( M or L ). Used with spi interfaces
txs	use select output \TSS in master mode ( Y or N ). Used with spi interfaces
type	Specify the type of shape to draw or the source of key data (touch or external)
U16	unsigned 16 bit integer data type
U32	unsigned 32 bit integer data type
U8	unsigned 8 bit integer data type
usbcom	usb com port
usbmsd	usb mass storage device
VAR	Variable having a certain style and a default value
VAR	Create a variable of a specified type with a default value eg VAR(Name,Value,Style)
WAIT	Wait specified milliseconds before next. eg WAIT(Time)
wdog	watchdog OFF(0), 100ms(100), 500ms(500), 1sec(1000)
width	Specify the border width of the shape



**Styles**

Styles enable you to maintain a common theme throughout your application and reduce the number of parameters required to be passed in the Page, text, draw, image and key commands. A style is only used during the creation of an entity. When updating a text or an image, the style is omitted from the command.

**Plan:** Style parameters can be updated using the dot operator except sizes and watchdog values.

LOAD(Txt32ASC.font,"ASCII8"); LOAD(Txt32ASC.rotate,varRotate); where varRotate holds 0,90,180 or 270.

Command	Description
<b>VAR(Name,Value,Style)</b>	<p><b>VAR Data Styles</b> Specify your own style for integer, float, pointer or text or use a built in style name STYLE(stVar, Data) {   type = U8;    // U8, U16, U32 - unsigned 8, 16 and 32 bit integer                 // S8, S16, S32 - signed 8, 16, 32 bit integer                 // TEXT for text strings                 // FLOAT for higher resolution calculation                 // POINTER for use with images   length=64;   // For text, specify the length from 1 to 8192, default =32   decimal=3;   // Specify the number of decimal places when type is float. Range 0 to 7, default=2   format="dd mm YY";   //Specify RTC format. see RTC page for format character types   location=SDRAM;    //Specify the data location as SDRAM (default) or EEPROM }</p> <p><b>Built In Styles (Add E for EEPROM types Example FLT4E)</b> The following pre defined 'built in' style names are available <b>U8/U8E</b>    - type = U8,   <b>U16/U16E</b> - type = U16,   <b>U32/U32E</b>   - type = U32 <b>S8/S8E</b>   - type = S8,   <b>S16/S16E</b> - type = S16,   <b>S32/S32E</b>   - type = S32 <b>PTR/PTR</b>   - type = pointer,   <b>TXT/TXTE</b> - type = TEXT, length=32 <b>FLT1/FLT1E</b> - type = float, decimal = 1,   <b>FLT2/FLT2E</b> - type = float, decimal = 2 <b>FLT3/FLT3E</b> - type = float, decimal = 3,   <b>FLT4/FLT4E</b> - type = float, decimal = 4</p> <p><b>Operational</b></p>
<b>PAGE(Name,Style) {.....}</b>	<p><b>Page Styles</b> The style defines the page size, position and background. STYLE(stPage,Page) //create a style name and define as type Page {   sizeX=480; //specify width of page 1 to 3* LCD width   sizeY=272; //specify height of page 1 to 3* LCD height   posX=0;    //specify the absolute X position of page on screen. -4 * LCD width to 4 * LCD width   posY=0;    //specify the absolute Y position of page on screen. -4 * LCD height to 4 * LCD height   back=black; //specify background colour of page as hex \000000 to \FFFFFF or colour name   image=pageimg; //specify background image of page as SDHC path or entity name using LIB. }</p> <p><b>Page with screen size or smaller.</b></p>
<b>TEXT(Name,Text,Style)</b>	<p><b>TEXT Styles</b> Fonts are available using single byte, 2 byte and UTF8 multi-byte coding. Built in ASCII fonts have the reserved names Ascii8, Ascii16, Ascii32 (case sensitive). Other library fonts are uploaded using the LIB command and have file type .FNT These are available for download from the character fonts web page at <a href="http://www.itrontft.com">www.itrontft.com</a>. <b>Unique Font Overlay</b> It is possible to overlay one font over another to enable single byte operation with ASCII from 20H to 7FH and Cyrillic, Greek, Hebrew, Bengali, Tamil, Thai or Katakana from 80H to FFH. The LIB command is used to load the extended font at 0080H instead of it's normal UNICODE location. The style for a text can then specify <b>font="MyASCII,MyThai"</b>; causing the Thai to overlap the ASCII from 80H to FFH.</p> <p>STYLE(Txt32ASC16,TEXT)    //assign a name for the style like Txt32ASC16 {   font="ASC16B,16THAI";    //define fonts using built in or preloaded .FNT files via LIB command   size=2;                  //a 24x24 font is expanded to a 48x48 font. default=1   col=white;                //"\000000" to "\FFFFFF" or reserved words from the colour chart.   maxLen=64;                //maximum length of text. default = 32, maximum=512   maxRows=4;                //maximum number of rows=32 where new line code \OD\OA is used.   rotate=90;                //rotation relative to screen 0, 90, 180, 270. default=0   curRel=CC;                //specify placement relative to cursor. CC Centre Centre , TC Top Centre,                               //BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left,                               // BL Bottom Left, TR Top Right, BR Bottom Right }</p> <p><b>Operational</b></p>
<b>DRAW(Name,X,Y,Style)</b>	<p>Draw or update a Line, Box or Circle of size X,Y or Pixel at X,Y. The entities can be an outline or filled.</p> <p><b>Draw Styles</b> It is possible to specify transparency values with colours if the colour is entered as a 32-bit hex number the top 8 bits specify the alpha blending level. col = \aarrggbb;   back = \aarrggbb;   where aa = alpha level. For example, col = \80FFFF00; gives 50% transparent yellow.</p> <p>STYLE(stCircleRed,DRAW) {   type=B;    //Specify the type of shape to draw. <b>type</b> = B or Box , C or Circle, L or Line, P or Pixel   col=red;    //Specify the border colour of the shape. Use hex, colour name + alpha   width=1;    //Specify the border width of the shape default = 1   back=\00FF66; //Specify the fill colour of the shape. Use hex, colour name + alpha   maxX=160;   // Declare the maximum width allowing for rotation   maxY=40;    // Declare the maximum height allowing for rotation   rotate=0;   // Specify the rotation of the shape with respect to the screen. 0,90,180,270   curRel=CC;   //specify placement relative to cursor. CC Centre Centre , TC Top Centre,                   //BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left,                   // BL Bottom Left, TR Top Right, BR Bottom Right }</p> <p><b>Operational</b></p>
<b>IMG(Name,Source,X,Y,Style)</b>	<p><b>Image Styles</b> The image may be larger than the size specified so it is necessary to define how it will be scaled. STYLE(MyImage,Image) {   scale=100;    // The image is scaled down or up by a percentage.                   //Supports 5% steps below 100 and 100% steps above 100.</p>

## iSMART Noritake Itron 5.7" TFT Module

	<pre> maxX=160; // Declare the maximum width allowing for rotation maxY=40; // Declare the maximum height allowing for rotation rotate=0; // Specify the rotation of the shape with respect to the screen. 0,90,180,270 curRel=CC; // specify placement relative to cursor. CC Centre Centre , TC Top Centre, } // BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left, // BL Bottom Left, TR Top Right, BR Bottom Right </pre> <p><b>Operational except scale C.</b></p>
<p><b>KEY(Name,Function,X,Y,Style)</b></p>	<p><b>KEY Styles</b> Specify the source of key data. If you require a dual action, specify 2 keys at the same location, one with action D and one with U.</p> <pre> STYLE(myTouch,key) { type=touch; //specify 'touch' screen or external 'keyio' debounce=250; //Specify the time delay to allow a key press to stabilise. Value in milliseconds. delay=1000; //Specify the time delay before auto repeat occurs. Value in milliseconds. 0=off. repeat=500; //Specify the repeat period if the key is held down. Value in milliseconds action = D; // Specify D or Down and U or Up. Specify the up or down action point for the key. curRel=CC; //specify touch key placement relative to cursor. CC Centre Centre , TC Top Centre, } //BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left, // BL Bottom Left, TR Top Right, BR Bottom Right </pre> <p><b>Operational for stylus. Touch with finger requires large key plus sampling parameter to be added.</b></p>

**Setup**

Setups for the interfaces are shown below with an explanation of the parameters.

Parameters can be updated using the dot operator

```
LOAD(RS4.baud,19200);
LOAD(RS4.proc,"CR");
```

Interface	Setup
<b>System</b>	<pre>setup(system) { bled=100;           //set backlight to OFF=0 or ON=100, 1-99 brightness levels available v4 PCB, v32                     // firmware wdog=100;          //set the watchdog time out period in milliseconds. rotate=0;          //set the rotation of the screen with respect to PCB test=showTouchAreas; //hide or show touch areas during product development calibrate=n;       //initialise the internal touch screen calibration screen. This automatically returns to                     // the previous page on completion. If it is necessary to abort then send                     //setup( system ) {calibrate=n}; encode=s;          //ASCII handling with extended unicode/utf8 in occasional strings } </pre>
<b>RS232</b>	<pre><b>Quick Setup</b> setup(RS2) { set="96NC" //quick set up combination "48, 96, 192, 384, 768, 1150 with parity N, O, E and Command            //option". }  <b>Setup</b> setup(RS2) { baud=38450; //num = 110 to 115200. Any value can be set to allow trimming for deviating clocks i.e. 38450 data=6;     //num = 5, 6, 7, 8 stop=15;   //num = 1, 15, 2 - note 15 is 1.5 bits parity=N;  //first letter of Odd, Even, None, Mark, Space rx=Y;      //set receive interface as active (Y), a command processing source (C) or disable (N). Default = N proc=",";  //process on receive termination character. See below procDel=Y; //remove or keep the termination character(s) before processing rxb=8246;  //set size of receive buffer in bytes. Default = 8192 bytes txi=Y;     //set transmit interface as active (Y), to echo command processing (E) or disable (N) txb=8350;  //set size of transmit buffer in bytes. Default = 8192 bytes encode=s;  //set s=ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw data bytes. flow=N;    //none, hardware RTS/CTS or DTR/DSR, software XON XOFF } </pre>
<b>RS485</b>	<pre><b>Quick Setup</b> setup(RS4) { set="96NC" //quick set up combination "48,96,192,384,768,1150 with parity N,O,E and Command option". }  <b>Setup</b> setup(RS4) { baud=38450; //num = 110 to 115200. Any value can be set to allow trimming for deviating clocks i.e. 38450 data=6;     //num = 5, 6, 7, 8 stop=15;   //num = 1, 15, 2 - note 15 is 1.5 bits parity=N;  //first letter of Odd, Even, None, Mark, Space rx=Y;      //set receive interface as active (Y), a command processing source (C) or disable (N). Default = N proc=",";  //process on receive termination character(s). See below procDel=Y; //remove or keep the termination character(s) before processing rxb=8196;  //set size of receive buffer in bytes. Default = 8192 bytes txi=Y;     //set transmit interface as active (Y), to echo command processing (E) or disable (N) txb=8196;  //set size of transmit buffer in bytes. Default = 8192 bytes encode=s;  //set s=ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw data bytes. flow=n;    //set n=none, s=software XON,XOFF } </pre>
<b>AS1, AS2, DBG</b>	<pre><b>Quick Setup</b> setup(AS1) //can setup AS1, AS2 or DBG { set="96NC" //quick set up combination "48,96,192,384,768,1150 with parity N, O, E and Command option". }  <b>Setup</b> setup(AS1) //can setup AS1, AS2 or DBG { baud=38450; //num = 110 to 115200. Any value can be set to allow trimming for deviating clocks i.e. 38450 data=7;     //num = 5, 6, 7, 8 stop=2;     //num = 1, 15, 2 - note 15 is 1.5 bits parity=N;  //first letter of Odd, Even, None, Mark, Space rx=Y;      //set receive buffer interface as active (Y), a command processing source (C) or disable (N).            //Default = N proc=",";  //process on receive termination character(s). See below procDel=Y; //remove or keep the termination character(s) before processing rxb=8246;  //set size of receive buffer in bytes. Default = 8192 bytes txi=Y;     //set transmit buffer interface as active (Y), to echo command processing (E) or disable (N) txb=8246;  //set size of transmit buffer in bytes. Default = 8192 bytes encode=s;  //set s=ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw data bytes. flow=N;    //none, hardware RTS/CTS or DTR/DSR, software XON XOFF } </pre>
<b>CANBUS Adaptor</b>	<pre>setup(AS1) { baud=38400; //num = 110 to 115200. Any value can be set to allow trimming for deviating clocks i.e. 38450 data=8;     //num = 5, 6, 7, 8 } </pre>

# iSMART Noritake Itron 5.7" TFT Module

	<pre> stop=1;          //num = 1, 15, 2 - note 15 is 1.5 bits parity=N;       //first letter of Odd, Even, None, Mark, Space rx=C;           //set receive buffer interface as active (Y), a command processing source (C) or disable (N).                 //Default = N encode=sr;      //set s=ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw data bytes. flow=H;         //none, hardware RTS/CTS or DTR/DSR, software XON XOFF } </pre>
<b>SPI</b>	<p><b>Quick Setup</b></p> <pre> setup(spi) { set="MR100";    //quick set up as Master/Slave, edge R/F, Command and speed 20-1000 } </pre> <p><b>Setup</b></p> <pre> setup(spi) { active=M;       //set as Master, Slave or None for both transmit and receive. Default = N edge=R;        //uses Rising or Falling clock edge. Default = R speed=100;     //set transmit speed value in kilobits/sec from 20 to 1000 for master mode. Default = 100 rx=Y;         //set receive buffer interface as active (Y), a command processing source (C) or disable (N).                 //Default = N proc="";      //process on receive termination character(s). See below. procDel=Y;    //remove or keep the termination character(s) before processing encode=s;     //set s=ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw data bytes. rxb=8264;    //set size of receive buffer in bytes. Default = 8192 bytes rxo=M;       //set receive data order as most significant bit (M) or least significant bit (L). Default = M rxf=N;       //use none or hardware MB to signify receive buffer full. Default = N rxs=N;       //use select input \RSS. Default = N txi=Y;       //set transmit buffer interface as active (Y), to echo command processing (E) or disable (N) end="nn"     //byte returned when no data left in display's spi transmit buffer and as a dummy byte to                 //send if required. txb=8244;    //set size of transmit buffer in bytes. Default = 8192 bytes txo=M;       //set transmit data order as most significant bit (M) or least significant bit (L). Default = M txf=N;       //none or hardware HB used to signify halt transmit in master mode. Default = N txs=N;       //use select output \TSS in master mode. Default = N } </pre>
<b>TWI / I2C</b>	<p><b>Quick Setup</b></p> <pre> setup(i2c) { set = "C7E";   //quick set up of I2C - Slave with Command and Address } </pre> <p><b>Setup</b></p> <pre> setup(i2c) { addr="3E";    //address pair where nn for write and nn+1 for read with range 02 to FE. end="\00";   //byte returned when no data left in display's i2c transmit buffer active=S;    //set as Master (M) or Slave (S) or disabled (N). Default = N speed=100;  //set transmit speed value in kilobits/sec from 20 to 400 for master mode. Default = 100 rx=Y;      //set receive buffer interface as active (Y), a command processing source (C) or disable (N).                 //Default = N proc="";   //process on receive termination character(s) procDel=Y; //remove or keep the termination character(s) before processing encode=s;  //s= ASCII single byte, w=UNICODE 2 byte, m=UTF8 multibyte rxb=8192; //set size of receive buffer in bytes. Default = 8192 bytes txi=Y;    //set transmit buffer interface as active (Y), to echo command processing (E) or disable (N) txb=8186; //set size of transmit buffer in bytes. Default = 8192 bytes } </pre>
<b>KEY I/O</b>	<pre> setup(keyio) { active=\0000FF; //high is active "\000000" &gt; "\FFFFFF", default is inactive inp=\00000C;   //high is input, low is output "\000000" &gt; "\FFFFFF" trig=\000001; //high is trigger interrupt "\000000" &gt; "\FFFFFF" as defined by edge edge=\000000; //high is rising edge, low is falling edge "\000000" &gt; "\FFFFFF" keyb=\000FF0; //high is scanned keyboard connection "\000000" &gt; "\FFFFFF" } </pre>
<b>PWM controller</b>	<pre> setup(pwm) { active=12;     //use 12 to synchronize PWM 1 and 2. N=none pol1=H;       //polarity = High or Low on first phase of PWM1 pol2=H;       //polarity = High or Low on first phase of PWM2 cycle1="200"; //cycle time in microseconds of PWM1. Range 160Hz to 1MHz cycle2 = "300"; //cycle time in microseconds of PWM2. Range 160Hz to 1MHz duty1= "44";  //value of first phase as a percentage for PWM1 = 1-99 duty2= "56";  //value of first phase as a percentage for PWM2 = 1-99 delay= "50";  //delay between first phase of PWM1 and first phase of PWM2 in microseconds } </pre>
<b>ADC - A to D converters</b>	<pre> setup( adc ) { active=12;    //set none, ADC1, ADC2 or both calib1=0.4;  //set value to use for calibration/scaling of ADC1 calib2=0.2;  //set value to use for calibration/scaling of ADC2 avg1=16;    //number of samples read and then averaged for ADC1 avg2=16     //number of samples read and then averaged for ADC2 } </pre>

Character Fonts



Compact Narrow Fonts	Wide Rounded Fonts
<a href="#">ASCII Base Page</a>	<a href="#">ASCII + European</a>
<a href="#">PC437 (USA - European Standard)</a>	<a href="#">Cyrillic</a>
<a href="#">PC850 (Multilingual)</a>	<a href="#">Greek</a>
<a href="#">PC852 (Latin 2)</a>	<a href="#">Arabic</a>
<a href="#">PC858 (Multilingual)</a>	<a href="#">Hebrew</a>
<a href="#">PC860 (Portuguese)</a>	<a href="#">Bengali</a>
<a href="#">PC863 (Canadian French)</a>	<a href="#">Tamil</a>
<a href="#">PC865 (Nordic)</a>	<a href="#">Thai</a>
<a href="#">PC866 (Cyrillic)</a>	<a href="#">Chinese/Japanese/Korean</a> TBA
<a href="#">WPC1252</a>	<a href="#">Hangul</a> TBA
<a href="#">Katakana</a>	<a href="#">Katakana</a>

You can include the character fonts required for an application by downloading the attached files and use the LIB command to store them in memory. You can setup your system to process text as single byte, 2 byte UNICODE or multibyte UTF8. See the LIB command for installing fonts. System fonts ASCII8, ASCII16 and ASCII32 are built in. The wide rounded fonts are preferred for higher quality designs.

It is possible to overlay one font over another to enable single byte operation with ASCII from 20H to 7FH and Cyrillic, Greek, Hebrew, Bengali, Tamil, Thai or Katakana from 80H to FFH. The LIB command is used to load the extended font at 0080H instead of it's normal UNICODE location. The style for a text can then specify font="MyASCII,MyThai"; causing the Thai to overlap the ASCII from 80H to FFH

Example

```
LIB( ascii24,"sdhc/asc_24.fnt"); //upload ascii 24 pixel wide font
LIB( cur24,"sdhc/cur_24.fnt?start=\\0080"); //upload currency font to 80H
```

In text style...

```
font="ascii24,cur24"; //cur24 overlays ascii24 at 80H-8FH
```

STANDARD ASCII - 20H to 7FH

Standard ASCII text in the range 20H to 7FH can be directly typed from the keyboard.

System fonts named ASCII8, ASCII16, ASCII32 are pre-installed.

```
Example TEXT( txt1, "Hello World", stTXT ); //single byte access to 20H to 7FH ASCII characters
```

EXTENDED ASCII - 20H to FFH

2/ When using single byte ASCII in the range 20H to 7FH, you can access extended characters from 80H to FFH using hex code like [\\AB](#)

```
Example TEXT( txt1, "1. A\\B0CDEF \\AB s", stTXT ); //single byte access to 80H to FFH
```

UNICODE and UTF8

3/ When using single byte ASCII in the range 20H to 7FH, you can access UNICODE characters by using hex code like [\\W0D7F](#) or a UTF8 character using hex code like [\\mC2AB](#). The symbols <...> are used where more than one character is coded.

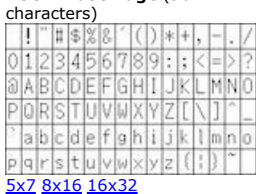
Examples

```
TEXT( txt2, "2. A\\w00B0CDEF \\w00AB", stTXT ); // UNICODE double byte access to 0080H to FFFFH
TEXT( txt3, "3. A\\mC2B0CDEF \\mC2AB", stTXT ); // UTF8 multi byte access to 80H to FFFFH
TEXT( txt5, "5. A\\sB0C\\w<00440045>F \\w00AB", stTXT ); // <...> are used for long hex strings \\s is used for single byte in a UNICODE or UTF8 encoded system
TEXT( txt7, "\\<372E204142B04344454620AB>", stTXT ); // string of single byte hex in the range 20H to 80H
TEXT( txt8, "\\w<0038002e00200041004200B0043004400450046002000AB>", stTXT );
TEXT( txt9, "\\m<392E204142C2B04344454620C2AB>", stTXT );
```

COMPACT NARROW FONTS (Single Byte Range 20H to FFH or UNICODE Range 0020H to 00FFH)

The ASCII base page is included automatically at 20H-7FH and the other fonts are automatically loaded to 80H to FFH. This gives a single byte range of 20H to FFH.

ASCII Base Page (96 characters)



5x7 8x16 16x32

PC437 (128 characters)



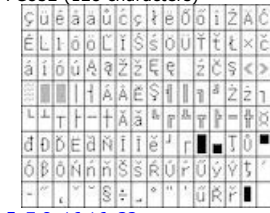
5x7 8x16 16x32

PC850 (128 characters)



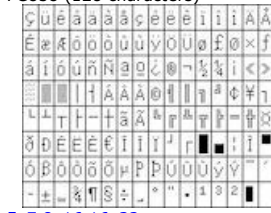
5x7 8x16 16x32

PC852 (128 characters)



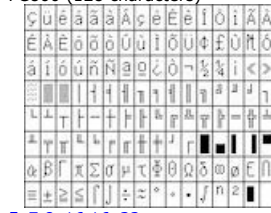
5x7 8x16 16x32

PC858 (128 characters)



5x7 8x16 16x32

PC860 (128 characters)



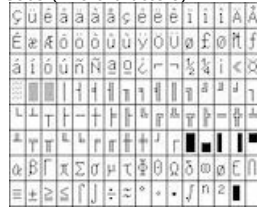
5x7 8x16 16x32

PC863 (128 characters)



[5x7 8x16 16x32](#)

PC865 (128 characters)



[5x7 8x16 16x32](#)

PC866 (128 characters)



[5x7 8x16 16x32](#)

WPC1252 (128 characters)



[5x7 8x16 16x32](#)

Katakana (128 characters)



[5x7 8x16 16x32](#)

**WIDE ROUNDED Fonts** (Single Byte Range 20H to FFH or UNICODE Range 0020H to FFFFH)

When loading these fonts into library, it is necessary to specify the offset address for the first character of each font table if a variation from UNICODE is required. The supplementary characters above FFFF are not supported in UTF8.

**ASCII + European (467 characters)**

A  
[16px \(3.2mm\)](#)

A  
[24px \(4.8mm\)](#)

A  
[32px \(6.4mm\)](#)

[40px \(8mm\)](#)

[48px \(9.6mm\)](#)

[60px \(12mm\)](#)

[72px \(14.4mm\)](#)

Unicode Range  
0020 - 0217

!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/	
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{		}	~	
ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı	ı
"	"	"	"	"	"	"	"	"	"	"	"	"	"	"	"
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	ß	À
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	ÿ	

**Cyrillic (226 characters)**

Ѥ  
[16px \(3.2mm\)](#)

Ѥ  
[24px \(4.8mm\)](#)

Ѥ  
[32px \(6.4mm\)](#)

Unicode Range  
0401 - 04F9

Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ	Ѱ	ѱ	Ѳ	ѳ
А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
ѣ	Ѥ	ѧ	Ѩ	ѩ	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ	Ѱ	ѱ	Ѳ	ѳ	Ѵ
Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ
Ѧ	ѧ	Ѩ	ѩ	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ
Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ
Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ
Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ
Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ
Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ



Tamil (61 characters)														
ஊ														
16px (3.2mm)	ஊ	ஈ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ
24px (4.8mm)	ஊ	ஈ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ
32px (6.4mm)	ஊ	ஈ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ
Unicode Range 0B82 - 0BF2	ஊ	ஈ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ	ஊ

Thai (87 characters)														
๘														
16px (3.2mm)	๘	๙	๐	๑	๒	๓	๔	๕	๖	๗	๘	๙	๐	๑
24px (4.8mm)	๘	๙	๐	๑	๒	๓	๔	๕	๖	๗	๘	๙	๐	๑
32px (6.4mm)	๘	๙	๐	๑	๒	๓	๔	๕	๖	๗	๘	๙	๐	๑
Unicode Range 0E01 - 0E5B	๘	๙	๐	๑	๒	๓	๔	๕	๖	๗	๘	๙	๐	๑

Chinese/Japanese/Korean (21151 characters) TBA														
挑	怀	恣	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒
16x16 (3.2mm)	挑	恣	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒
24x24 (4.8mm)	挑	恣	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒
32x32 (6.4mm)	挑	恣	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒
Unicode Range 3300 - 9FA5	挑	恣	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒	忒

Hangul (11172 characters) TBA														
넛	체	체	체	체	체	체	체	체	체	체	체	체	체	체
16x16 (3.2mm)	넛	체	체	체	체	체	체	체	체	체	체	체	체	체
24x24 (4.8mm)	넛	체	체	체	체	체	체	체	체	체	체	체	체	체
32x32 (6.4mm)	넛	체	체	체	체	체	체	체	체	체	체	체	체	체
Unicode Range AC00 - D7A3	넛	체	체	체	체	체	체	체	체	체	체	체	체	체

Katakana (94 characters)														
ポ	ア	アイ	ウ	エ	エ	オ	カ	ガ	キ	ギ	ク			
16x16 (3.2mm)	ポ	グ	ケ	ゴ	サ	ザ	シ	ジ	ス	ズ	セ	ゼ	ソ	ゾ
24x24 (4.8mm)	ポ	ダ	チ	ツ	ツ	ツ	テ	デ	ト	ド	ナ	ニ	ヌ	ノ
32x32 (6.4mm)	ポ	ハ	パ	ヒ	ピ	フ	ブ	ヘ	ベ	ペ	ホ	ボ	ポ	マ
Unicode Range 30A1 - 30FE	ポ	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ヰ



**Colour Chart**

The colour chart below shows the built in colours of the TFT module. To clarify the reference name of a colour, hover over the hex code.

#4682B4 steelblue	#041690 royalblue	#6495ED cornflowerblue	#B0C4DE lightsteelblue	#7B68EE mediumslateblue	#6A5ACD slateblue	#483D8B darkslateblue	#191970 midnightblue	#000080 navy	#00008B darkblue
#0000CD mediumblue	#0000FF blue	#1E90FF dodgerblue	#00BFFF deepskyblue	#87CEFA lightskyblue	#87CEEB skyblue	#ADD8E6 lightblue	#B0E0E6 powderblue	#F0FFFF azure	#E0FFFF lightcyan
#AFEEEE paleturquoise	#48D1CC mediumturquoise	#20B2AA lightseagreen	#008B8B darkcyan	#008080 teal	#5F9EA0 cadetblue	#00CED1 darkturquoise	#00FFFF aqua	#00FFFF cyan	#40E0D0 turquoise
#7FFFD4 aquamarine	#66CDAA mediumaquamarine	#8FBC8F darkseagreen	#3CB371 mediumseagreen	#2E8B57 seagreen	#006400 darkgreen	#008000 green	#228B22 forestgreen	#32CD32 limegreen	#00FF00 lime
#7FFF00 chartreuse	#7FCF00 lawngreen	#ADFF2F greenyellow	#9ACD32 yellowgreen	#98FB98 palegreen	#90EE90 lightgreen	#00FF7F springgreen	#00FA9A mediumspringgreen	#556B2F darkolivegreen	#6B8E23 olivedrab
#808000 olive	#BDB76B darkkhaki	#B8860B darkgoldenrod	#DAA520 goldenrod	#FFD700 gold	#FFFF00 yellow	#F0E68C khaki	#EEE8AA palegoldenrod	#FFB6C1 blanchedalmond	#FFA07A moccasin
#F5DEB3 wheat	#FFDEAD navajowhite	#DEB887 burlywood	#D2B48C tan	#BC8F8F rosybrown	#A0522D sienna	#8B4513 saddlebrown	#D2691E chocolate	#CD853F peru	#F4A460 sandybrown
#8B0000 darkred	#800000 maroon	#A52A2A brown	#B22222 firebrick	#CD5C5C indianred	#F08080 lightcoral	#FA8072 salmon	#E9967A darksalmon	#FFA07A lightsalmon	#FF7F50 coral
#FF6347 tomato	#FF8C00 darkorange	#FFA500 orange	#FF4500 orangered	#DC143C crimson	#FF0000 red	#FF1493 deeppink	#FF00FF fuchsia	#FF00FF magenta	#FF69B4 hotpink
#FFB6C1 lightpink	#FFC0CB pink	#DB7093 palevioletred	#C71585 mediumvioletred	#800080 purple	#8B008B darkmagenta	#9370DB mediumpurple	#8A2BE2 blueviolet	#4B0082 indigo	#9400D3 darkviolet
#9932CC darkorchid	#BA55D3 mediumorchid	#DA70D6 orchid	#EE82EE violet	#DDA0DD plum	#D8BFD8 thistle	#E6E6FA lavender	#F8F8FF ghostwhite	#F0F8FF aliceblue	#F5FFFA mintcream
#F0FFF0 honeydew	#FAFAD2 lightgoldenrodyellow	#FFFACD lemonchiffon	#FFF8DC cornsilk	#FFFFE0 lightyellow	#FFFFFF ivory	#FFF5F5 floralwhite	#FAF0E6 linen	#FDF5E6 oldlace	#FAEBD7 antiquewhite
#FFE4C4 bisque	#FFDAB9 peachpuff	#FFEFD5 papayawhip	#F5F5DC beige	#FFF5EE seashell	#FFF0F5 lavenderblush	#FFE4E1 mistyrose	#FFF5F5 snow	#FFFFFF white	#F5F5F5 whitesmoke
#DCDCDC gainsboro	#D3D3D3 lightgrey	#C0C0C0 silver	#A9A9A9 darkgray	#808080 gray	#778899 lightslategray	#708090 slategray	#696969 dimgray	#2F4F4F darkslategray	#000000 black

**Getting Stared with iSMART TFTs**

If you received a development kit with USB cable and SD card inserted into a xxx-K612A1TU module, just plug in the USB cable between a PC and the display module. The boot code and operational software will load and then run the file TU480A.mnu from the SD card.

The supplied demonstration sequences through 4 screens. The elevator and aircon screens are working applications so you can press the touch keyboard to operate. After 20 seconds of inaction, the demonstration moves on to the next screen.

After experimenting with the demonstration, review the basic applications below. Do not hesitate to send us an email for further explanation. Key issues to understand..

- 1/ The system uses text commands rather than difficult to remember hex codes.
- 2/ All objects and functions are given a name for easy future referencing.  
Interfaces are given pre-defined names like RS2 for RS232 and RS4 for RS485.
- 3/ Commonly used parameters are stored in 'styles' like in HTML web pages.  
This reduces the number of commands from 250 in a conventional TFT module to just 25 in iSMART TFTs with equal or better functionality.

A typical menu file's commands will be constructed and ordered as follows (detail removed for clarity):

```
LIB...           //load in images and fonts from memory into library
LIB...
INC..           //include another menu file which may have global styles and setup.

STYLE...       //define styles for pages, text, images used in this file
STYLE...

SETUP..        //setup system and external interfaces like RS232
SETUP...

VAR...         //create variables used for calculation, temporary storage and pointing
VAR...

PAGE(MAIN,styleMain) { //create a main page with text, images and associated keys
  POSN... TEXT      //place text at a specified position on screen
  POSN... IMG       //place icon / image at a specified position on screen
  POSN... KEY       //place a touch key area on screen and define function to call
}

PAGE(SUB,stylePage1) { //create other pages
  POSN... TEXT      //place text at a specified position on screen
  POSN... IMG       //place icon / image at a specified position on screen
  .....
  LOOP(CntLoop,FOREVER) { IF(CNTMINS=0,FncZero); } // function calls associated with page
}

FUNC(FncZero) { LOAD(RS2,"Hour Count = ",CNTHRS,"\\0A\\0D"); //send message to host via RS232
FUNC(MyFunc) { .....} //other functions associated with key press or interfaces

INT...         // Initialise interrupts for slave timers and inputs...not host interface - use setup with v39 software

SHOW(MAIN); // After pre-loading all style parameters, pages and functions, start the application with first page.
After this point, functionality follows page key presses and functions or incoming command data from host or interfaces.
```

When creating an entity for the first time, include the style parameter. To update the entity omit the style parameter. If you specify the style again, you will create a copy.

Entities are layered on the screen from back to front in the order they are listed in the menu with the screen background defined in the page style. If you want a button image to change colour, include one colour button in your background and the other colour button as a separate image over the top. To change colour, just HIDE and SHOW the top button. This technique is used in the air conditioner project.

The examples below can be cut and pasted from their box into a text editor (NotePad). Save the file as TU480A.mnu and copy onto the SD card. Plug it into the iSMART TFT module, apply power and view the result.

**Hello World from Internal Menu**

```
// Menu file TU480A.MNU for Demo using TU480X272C and v32 firmware update
// Simple demo to display text

STYLE(BlackPg, Page) { Back=black;} //black background
STYLE( Txt32White, Text )
{
font=Ascii32; col=white; maxLen=32; maxRows=1; curRel=CC; //white system text 32 pixels high
}

PAGE( MainPg, BlackPg )
{
  POSN( 240, 136 ); // Set writing position to centre of display
  TEXT( Text1, "Hello World", Txt32White ); // Draw text
}

SHOW( MainPg );
//end
```

**Hello World via RS232 IN with touch key to send RS232 OUT**

```
// Menu file TU480A.MNU for Demo using TU480X272C
// 07-Oct-2010
// This example is identical to example 1 except RS232 is defined
// using setup for command mode at19200 baud, no parity

STYLE(BlackPg, Page) //define page style
{
Back=black; //background is black
}

STYLE( Txt32White, Text ) //define text style
{
font=Ascii32; //use built in font
col=white; //text colour is white
maxLen=32;
maxRows=1;
curRel=CC; //centre position
}

VAR(mytxtVar,"Hello People",TXT); //create a text variable to hold up to 32 characters

PAGE( MainPg, BlackPg )
{
POSTN( 240, 136 ); // Set writing position to centre of display
TEXT( Txt1, mytxtVar, Txt32White ); // Create text area at the writing position
KEY(Key1,[LOAD(RS2,mytxtVar,"\\0D\\0A");],470,270,TOUCH); //Touch screen to sends content of mytxtVar plus CRLF out of RS232 port
}

SETUP( RS2 )
{
set = "192NC"; // 19200 bps, no parity, command mode
}

SHOW( MainPg );

// Send text command to the display via RS232 : LOAD( mytxtVar, "Hello World" );;\0D
// Note :-
// Sending 2 semicolons is equivalent to SHOW (currentpage);
// All command lines must be followed by CR (\0D)
//If your system can send binary \00D can be sent as 0DH
```

**Images loaded, flashed and moved**

```
// Menu file TU480A.MNU for Demo using TU480X272C
// 11-Oct-2010
// This example places 2 images on the display with one flashed and moved.

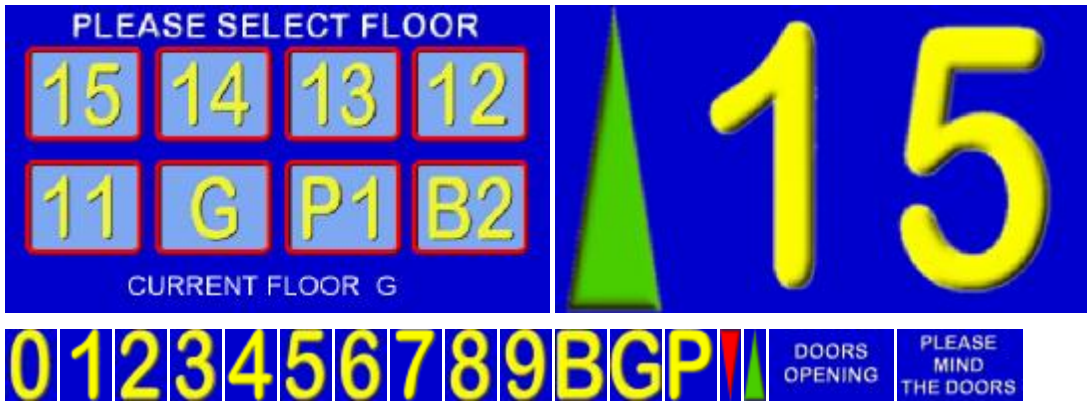
LIB(Image1,"SDHC/lift1.bmp?back=\\0000CD"); //load image1 from SD card
LIB(Image2,"SDHC/lift2.bmp?back=\\0000CD"); //load image2 from SD card
STYLE(BluePg, Page) {back=\\0000CD;}; //define style of page with blue background
STYLE(StImg, Image) {curRel=CC;}; //centre image with respect to POSN cursor

PAGE( MainPg, BluePg )
{
POSTN( 199, 136 ); IMG( LeftImg, Image1,172,240,StImg); // Position and draw 1st image on display
POSTN( 396, 136 ); IMG( RightImg, Image2,172,240,StImg); // Position and draw 2nd image on the display
}

SHOW( MainPg ); //show page

WAIT(2000); //wait 2 seconds
HIDE( LeftImg ); //hide left image and refresh page
WAIT(2000);
SHOW( LeftImg ); //show left image and refresh page
WAIT(2000);
POSTN( 396,136,LeftImg); //position left image under right image and refresh page
// Sending 2 semicolons is equivalent to SHOW (currentpage);
//You will see a blue border around the right image due to background transparency differences.
```

## Elevator Control System



The user can select a floor and travel from any floor to another floor.  
 The arrow is selected according to direction.  
 Warning signs for doors opening and closing are used.  
 Variables are used to store the current floor and destination floor.  
 An RS232 interface could be added to communicate with other floor indicators.  
 Download Image Files from website

## Elevator System Code using V30+ software

```
(Highlight, cut and paste below this line)
// Menu file TU640A.MNU for Elevator System using TU640X480C
// Updated 06-Nov-2010
// Floors are 15(15)..01(1), G(0), P1(-1), B2(-2)

// Load images into the library
LIB(libImgNum0,"SDHC/Lift06.bmp?back=\\0000CD"); // Load Number 0
LIB(libImgNum1,"SDHC/Lift16.bmp?back=\\0000CD"); // Load Number 1
LIB(libImgNum2,"SDHC/Lift26.bmp?back=\\0000CD"); // Load Number 2
LIB(libImgNum3,"SDHC/Lift36.bmp?back=\\0000CD"); // Load Number 3
LIB(libImgNum4,"SDHC/Lift46.bmp?back=\\0000CD"); // Load Number 4
LIB(libImgNum5,"SDHC/Lift56.bmp?back=\\0000CD"); // Load Number 5
LIB(libImgNum6,"SDHC/Lift66.bmp?back=\\0000CD"); // Load Number 6
LIB(libImgNum7,"SDHC/Lift76.bmp?back=\\0000CD"); // Load Number 7
LIB(libImgNum8,"SDHC/Lift86.bmp?back=\\0000CD"); // Load Number 8
LIB(libImgNum9,"SDHC/Lift96.bmp?back=\\0000CD"); // Load Number 9
LIB(libImgBChar,"SDHC/LiftB6.bmp?back=\\0000CD"); // Load Character B
LIB(libImgGChar,"SDHC/LiftG6.bmp?back=\\0000CD"); // Load Character G
LIB(libImgPChar,"SDHC/LiftP6.bmp?back=\\0000CD"); // Load Character P
LIB(libImgDTri,"SDHC/LiftDn6.bmp?back=\\0000CD"); // Load red triangle
LIB(libImgUTri,"SDHC/LiftUp6.bmp?back=\\0000CD"); // Load green triangle
LIB(libImgPMTD,"SDHC/LiftCls6.bmp"); // Load the warning message
LIB(libImgSelFlr,"SDHC/LiftSel6.bmp"); //Load the Select Floor Page
LIB(libImgDoors,"SDHC/LiftOpn6.bmp"); // Load the Doors Page

// Create styles
STYLE(stLiftPg,Page){back=\\0000CD;}
STYLE(stLiftMainPg,Page){back=\\0000CD;image=libImgSelFlr;}
STYLE(stGenImg,Image) {curRel=CC;}

LIB( fntAscii48, "SDHC/asc_48.fnt" ); // Load Ascii Font 48
STYLE(stTxt48W,Text) {font=fntAscii48; col=white;}

// Create vars
VAR(vS8,0,S8);
VAR(ptrLiftArr>"libImgUTri",PTR);
VAR(ptrLiftTens>"libImgGChar",PTR);
VAR(ptrLiftOnes>"libImgNum1",PTR);
VAR(vReqd,0,S8);
VAR(vThis,0,S8);
VAR(vMove,0,U8);

VAR( varRunDemo, 0, U8 );
VAR( varSecCnt, 0, U8 );
VAR( varCnt2, 1, U8 );
VAR( varDemoNum, 0, U8 );

// Create Select Floor Page
PAGE(pgLiftMain,stLiftMainPg){
  POSN(88,161); KEY(keyFlr15,[LOAD(vReqd,15);TEXT(txtCurFlr,"15");RUN(fncGo);],105,95,TOUCH);
  POSN(+156,+0); KEY(keyFlr14,[LOAD(vReqd,14);TEXT(txtCurFlr,"14");RUN(fncGo);],105,95,TOUCH);
  POSN(+156,+0); KEY(keyFlr13,[LOAD(vReqd,13);TEXT(txtCurFlr,"13");RUN(fncGo);],105,95,TOUCH);
  POSN(+156,+0); KEY(keyFlr12,[LOAD(vReqd,12);TEXT(txtCurFlr,"12");RUN(fncGo);],105,95,TOUCH);
  POSN(88,306); KEY(keyFlr11,[LOAD(vReqd,11);TEXT(txtCurFlr,"11");RUN(fncGo);],105,95,TOUCH);
  POSN(+156,+0); KEY(keyFlrG, [LOAD(vReqd, 0);TEXT(txtCurFlr, "G");RUN(fncGo);],105,95,TOUCH);
  POSN(+156,+0); KEY(keyFlrP1,[LOAD(vReqd,-1);TEXT(txtCurFlr,"P1");RUN(fncGo);],105,95,TOUCH);
  POSN(+156,+0); KEY(keyFlrB2,[LOAD(vReqd,-2);TEXT(txtCurFlr,"B2");RUN(fncGo);],105,95,TOUCH);

  POSN(220,430);TEXT(txtCurFlrLbl,"CURRENT FLOOR",stTxt48W);
  POSN(465,+0); TEXT(txtCurFlr,"G",stTxt48W);
  LOOP(lpLiftMain,FOREVER){RUN(fncDemoUpdate);}
}
```

## iSMART Noritake Itron 5.7" TFT Module

```
FUNC( fncDemoUpdate ) { IF( varRunDemo == 1 ? fncDemoUpdate2 ); } // Call from each demo
FUNC( fncDemoPause ) { LOAD( varCnt2, 20 ); } // Call from demo to pause change

FUNC( fncDemoUpdate2 ) { IF( CNTSECS != varSecCnt ? fncSecTimer ); }
FUNC( fncSecTimer ) { LOAD( varSecCnt, CNTSECS ); CALC( varCnt2, varCnt2, 1, "-" ); IF ( varCnt2 == 0 ? fncNextDemo ); }
FUNC( fncNextDemo )
{
  LOAD( varCnt2, 5 );
  CALC( varDemoNum, varDemoNum, 1, "+" );
  CALC( varDemoNum, varDemoNum, 4, "%" ); // Num Demo Screens
  IF( varDemoNum == 0 ? fncInfo );
  IF( varDemoNum == 1 ? fncLift );
  IF( varDemoNum == 2 ? fncAirCon );
  IF( varDemoNum == 3 ? fncTennis );
  IF( varDemoNum == 4 ? fncFonts );
}

// Level indication page
PAGE(pgIND,stLiftPg)
{
  POSN(90,240);IMG(imgTri,ptrLiftArr,86,200,stGenImg);HIDE(imgTri);
  POSN(299,+0);IMG(img10s,ptrLiftTens,172,240,stGenImg);
  POSN(496,+0);IMG(img1s,ptrLiftOnes,172,240,stGenImg);
  LOOP(lpLiftInd,FOREVER){IF(vMove=1?fncMove);}
}

// Lift is moving
FUNC(fncMove){
  IF(vThis>vReqd?[LOAD(ptrLiftArr>"libImgDTri");IMG(imgTri,ptrLiftArr);SHOW(imgTri);RUN(fncShowFlr);CALC(vThis,vThis,1,"-");]);
  IF(vThis<vReqd?[LOAD(ptrLiftArr>"libImgUTri");IMG(imgTri,ptrLiftArr);SHOW(imgTri);RUN(fncShowFlr);CALC(vThis,vThis,1,"+");]);
  IF(vThis=vReqd?[LOAD(vMove,0);HIDE(imgTri);RUN(fncShowFlr);RUN(fncDoorOpen);SHOW(pgLiftMain);]);
}

// Start lift moving
FUNC(fncGo){RUN(fncDemoPause);LOAD(vMove,1);HIDE(imgTri);RUN(fncDoorClose);RUN(fncShowFlr);}

// Show Current Floor
FUNC(fncShowFlr)
{
  IF(vThis>0?[CALC(vS8,vThis,10,"/");LOAD(ptrLiftTens>"libImgNum",vS8);CALC(vS8,vThis,10,"%");LOAD(ptrLiftOnes>"libImgNum",vS8);SHOW
  (img10s,img1s);]);
  IF(vThis=0?[LOAD(ptrLiftTens>"libImgGChar");SHOW(img10s);HIDE(img1s);]);
  IF(vThis=-1?[LOAD(ptrLiftTens>"libImgPChar");LOAD(ptrLiftOnes>"libImgNum1");SHOW(img10s,img1s);]);
  IF(vThis=-2?[LOAD(ptrLiftTens>"libImgBChar");LOAD(ptrLiftOnes>"libImgNum2");SHOW(img10s,img1s);]);
  IMG(img10s,ptrLiftTens);IMG(img1s,ptrLiftOnes);
  SHOW(pgIND);
  WAIT(1000);
}

// Create Door Closing and Opening
FUNC(fncDoorClose){SHOW(pgShut);WAIT(400);SHOW(pgBlnk);WAIT(100);SHOW(pgShut);WAIT(400);SHOW(pgBlnk);WAIT(100);SHOW
(pgShut);WAIT(400);SHOW(pgBlnk);WAIT(100);}
FUNC(fncDoorOpen){SHOW(pgOpen);WAIT(400);SHOW(pgBlnk);WAIT(100);SHOW(pgOpen);WAIT(400);SHOW(pgBlnk);WAIT(100);SHOW
(pgOpen);WAIT(400);SHOW(pgBlnk);WAIT(100);}

// Create Door Closing / Opening / Blank Pages
PAGE(pgShut,stLiftPg){POSN(319,239);IMG(imgDC,libImgPMTD,640,480,stGenImg);}
PAGE(pgOpen,stLiftPg){POSN(319,239);IMG(imgDO,libImgDoors,640,480,stGenImg);}
PAGE(pgBlnk,stLiftPg){}

// RUN Main page
SHOW(pgLiftMain);
```

**Frequently Asked Questions**

Please send your questions to [tech@noritake-itron.com](mailto:tech@noritake-itron.com)  
We will try to respond within 24 hours (Monday to Thursday)

Product Status and Availability  
Start Programming  
Memory and SDCard  
Interfacing  
Others

**PRODUCT STATUS and AVAILABILITY**

- What is the availability of this product ?

TU480X272C-K6121TU prototypes have been supplied to 122 companies in Europe from May 2010 **Yes**  
The planned European market release: 16th September 2010 **Yes**  
4.3inch available from the 5th October 2010 in volume. **Yes**  
400 evaluation units of 4.3inch sold by end of Dec 2010.  
Build 1600pcs in Q1/Q2 2011. **Kitted awaiting production.**  
5.7 inch and 7inch prototypes will be available **20th December. Yes**  
3.5inch prototypes will be available in March 2011.  
Linux operating system June 2011.

- What is the firmware development status of this product ?

We thank the 20 companies in Europe who helped beta testing  
As at 12th Dec, 96% of the specified software functionality complete.  
All existing customers are updated by email with latest release.  
Customer identified bug fix is implemented in the next release (7 -14 days).  
USB interfacing, arrays and structures are the main features still in development.  
Future developments include automated animation plus audio and video processing.

**START PROGRAMMING**

- What is needed to start a development with this TFT? Need SW-Development, HW Development-Kit ?

Hardware: TFT module plus SDCard plus SD Card Reader  
Software: PC with text editor (notepad) and image editor (paint or paint-shop) or download free iDevTFT software.  
The main objective of this product was to avoid expensive development tools for the customer.  
At this time the TU480A.mnu file on the SDCard must be edited externally using the supplied adaptor via a reader.  
The USB cable is used only for easy supply of power.  
When the SDCard is removed, the PC may ask for a USB driver. Press CANCEL to this request.  
Once USB has been developed, this will allow the user to edit files and send via the USB.  
1GB SD and SDHC 4G, 8G, 16G or 32G card can be used for im480c1.bin or other files.  
2G SD not supported.

- Can we program directly via the usb port into the flash ?

At the moment it is necessary to program and insert the SD card or send commands by async port.  
The plan for USB is behind schedule but estimated from end Jan 2011  
Production programming will then be possible via SDCard, serial port or USB.  
The FPROG command will then transfer to onboard NAND flash for future power on initialisation without SD card.

- Is the TFT interpreting a text based program language without the user compiling the software?

This is correct, although the TFT module internally converts the data to achieve the necessary speed.  
The uploaded text menu files are discarded for security.

-Is it possible to do screen captures for manuals or to run the interface on a PC as a simulator?

We supply iDevTFT for editing, local debugging and eventually PC simulation.

- Can we find a complete example?

Users can download examples from the web and the module can be supplied with pre-loaded applications.  
We are showing an Aircon, Elevator Systems and Keyboard at this time, with others available soon.  
These are updated as the firmware permits.

- Is the module available with Linux operating system?

We understand that some Linux has been written for this processor.  
Work on a Linux solution is scheduled for mid 2011.  
There are support issues for Linux.

- Why create a new operating system?

iSMART is a **combined** communications language and operating language.  
This gives it unique capabilities not possible in other languages.  
The compact command set is much easier to learn than other systems with 300 commands.

**MEMORY and SD CARD**

- Is it possible to download the text based program into Flash memory and then the TFT can be driven without SD-Card?

Yes, the SDCard is used as a convenient way of programming the module.  
Like a CD on PC, you do not need the CD after installing a program.  
The user can use instruction FPROG to store their program and images in FLASH.  
Check the release for function FPROG for availability.

- The specification says that user code is generated at Noritake – will this be changed in the future?

The menu and function code is generated by the customer.  
If the customer wants special functionality, we can produce a custom user software or (user code).  
Some requests have been included as features in the main software.

- Can the module perform as a data logger via the SDCard ?

Yes, this is being developed for SD card only. EEPROM can store up to 50 variables only as standard.

# iSMART Noritake Itron 5.7" TFT Module

SD Card is the preferred method since customer can replace the card easily.  
Customised firmware for writing to onboard 128M flash will be available.

-----  
-GUI systems usually require dynamic memory (malloc and free), which has to be handled very carefully in real-time systems. Are you using dynamic memory and if so, what system are you using?  
The freeing of memory is a key issue.  
We do have an equivalent to malloc and free.  
Entities are created in the library and data area assigned.  
When a DEL command (free) is used, the entry is marked for deletion however the actual clean up is under user control using the RESET(DELETED) command.

## INTERFACING

-How many RS232 serial ports can this support?  
There are 4 async ports as standard  
A CMOS level 4 line ON CN3 and 2 line on CN6 and CN12 where present.  
RS232 4 line on CN1 (includes DTR/DSR or CTS/RTS selection)  
The K611A version modules include an additional full duplex RS485 port on CN1 which is common with CN12.

-----  
-What sort of USB is fitted - host or device?  
It is planned to have a CDC Device (COM Port).  
Access to NAND will be write only for customer IPR security.  
HID Print and Keyboard will also be available later.

## OTHER

- What is the response time of a keystroke?  
The touch screen has 2 modes of operation.  
1/ Calculate key on press or release allowing for defined debounce.  
2/ Auto repeat after a defined period (delay) and resent until released (repeat).  
It is possible to define a function to run on key down or/and key up.  
This can be sent to a host interface or used internally for data entry or navigation.

-----  
-How is language support handled - is it possible to switch sets of strings for different languages?  
Strings can be declared as variables and pointed to any text object.  
Text data is internally processed and stored in UNICODE even though just ASCII may be used.

-On the current product we use a hybrid character set that is mostly western European but includes some central European characters, so we can use the same fonts for Polish.  
How would different character sets be handled?  
Unicode fonts are available for 90% of the world's languages.  
You can request selected additions for Eu50.  
The user can create their own image fonts on a paint software and reference them to UNICODE values

-The application slows down as each minute passes and after ~16mins the system crashes.  
A STYLE is only used during the first creation of a TEXT, DRAW or IMAGE command.  
When updating in a LOOP, omit the style other wise a new entity is created.  
Every entity remembers it's position and parameters

//Incorrect --- constantly creates new text entities

```
PAGE( Test, ST_Main )
{
  LOOP(animation,FOREVER)
  {
    POSN(100,100); TEXT(mins,CNTMINS,STtext24Yw);
    POSN(150,100); TEXT(secs,CNTSECS,STtext24Yw);
  }
}
```

//Correct --- creates text entities once then updates

```
PAGE( Test, ST_Main )
{
  POSN(100,100); TEXT(mins,CNTMINS,STtext24Yw);
  POSN(150,100); TEXT(secs,CNTSECS,STtext24Yw);
  LOOP(animation,FOREVER)
  {
    TEXT(mins,CNTMINS); //style omitted
    TEXT(secs,CNTSECS); // refresh page
  }
}
```