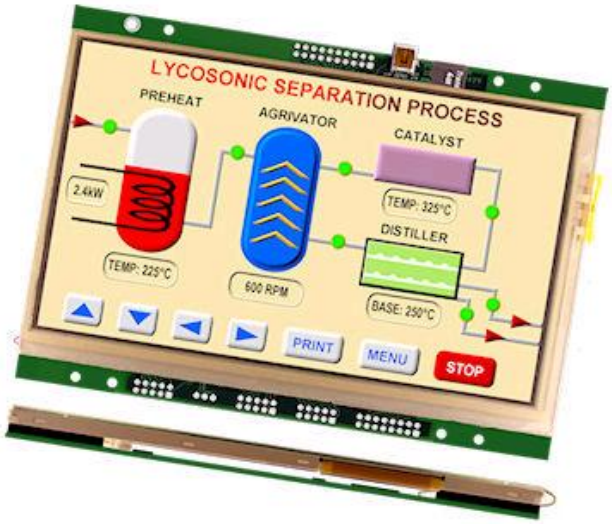


Noritake itron®

Product No	**TU800x480C-XXX**
Issue Date	**30/7/2012**
Document Ref	**42782**

General - 1
Basket

7.0" itron SMART TFT Module

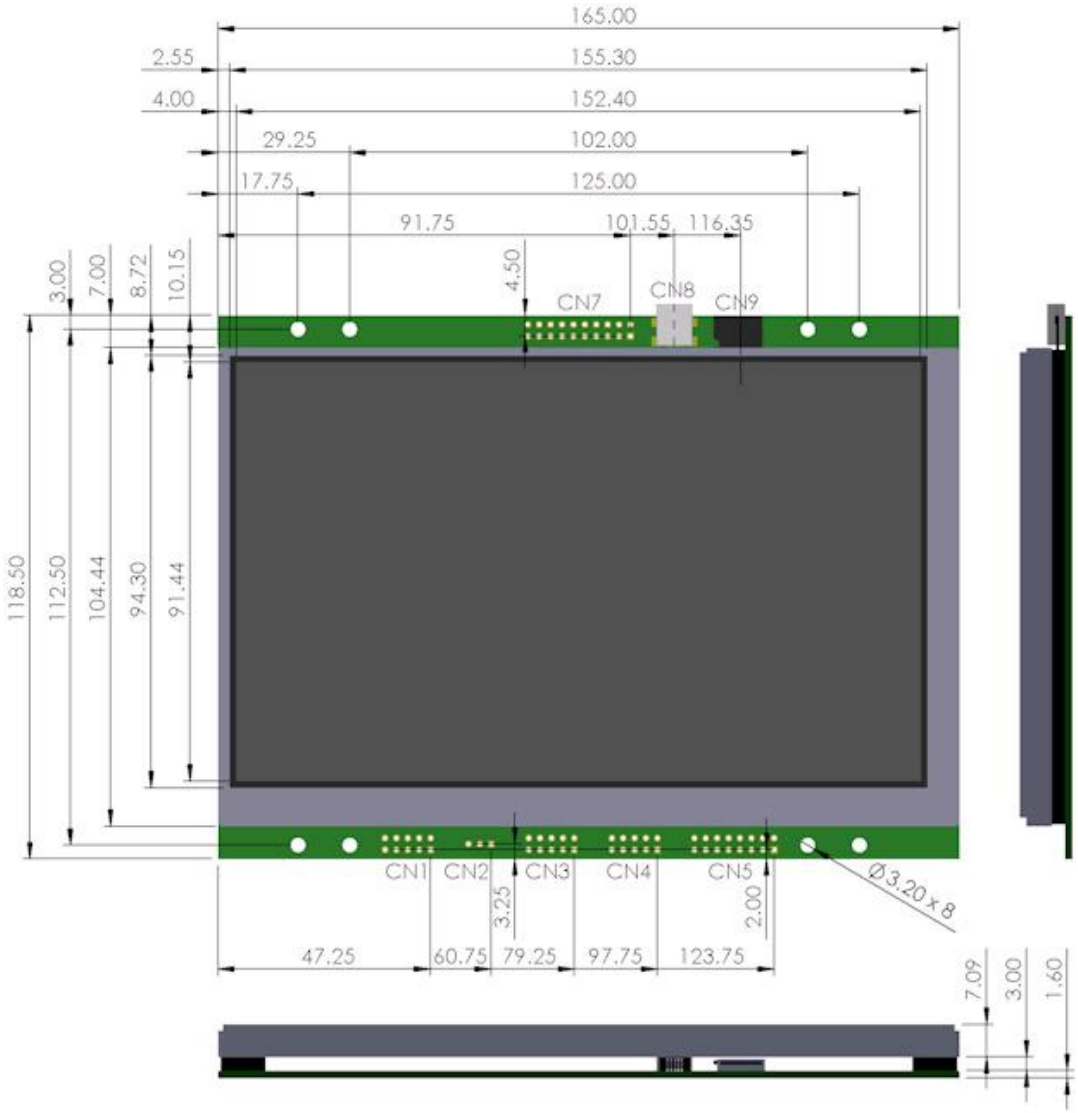
800x480 pixels
262,144 Colours (18 bit)
40 Page Display RAM
128M Byte Flash
4G+ Micro SDHC Slot
LED Backlight Control
5V Supply 3.3V Logic
ASCII + UNICODE Fonts

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
Object Oriented Language
Graphic User Interface
Integrated Debugger

Downloads
Full Spec (pdf)
Full Spec (compiled)
2D Mechanical
Environmental

The 7.09mm TFT thickness includes a touch screen. This dimension is reduced for non touch versions



Part Number Structure

T U 8 0 0 X 4 8 0 C - K 6 1 X A 1 T U B C L

- Pixels X x Y
- RS485 & RS485 1
- RS232 2
- LINUX
- CANBUS
- Battery Holder
- USB Socket
- Touch Panel

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 33 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.

TFT Module and Kit Options

Module Part Number	CN1 Interface	+ USB	+ TOUCH & USB	+ LINUX	Accessories
TU800x480C + suffix	RS232	**-K612A1U**	**-K612A1TU**	**-K612A1TUL**	View
	RS232 + RS485	**-K611A1U**	**-K611A1TU**	**-K611A1TUL**	

Hover over cart to view unit kit price which excludes freight and VAT. Modules do not include SD cards or cables. Option to pay in GBP £.

Optical & Environmental Parameters

Screen Type	800x480 pixels - RGB Stripe - Pixel Pitch 0.19x0.19mm
Display Area	152x91mm - 7.0" 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	6 o'clock (12 o'clock-Invert the PCB and set 180 degrees orientation in software)
Weight	261g including touch screen
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^2 (40G) 9mS X-Y-Z, 3 times each direction (Non operating)

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	580	620	650	mADC	VCC=5V	All dots on
Power Supply Current 2		ICC2	250	270	320	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
CN3	I2C Serial Mode	5V	SCL	-	SDA	0V	/IRQ	-	/RES	MB	HB	3v3 Logic (5v in option)
	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
	User I/O	5V	K24	K25	K26	0V	K27	K28	/RES	K29	K30	Additional I/O
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

NOTE: CN2 is the preferred power supply input and other supply connections used for powering peripherals

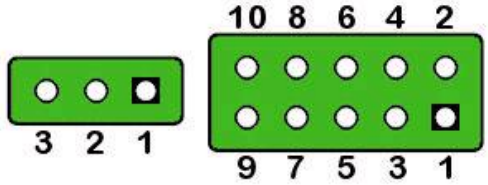
CON	Function	1/2	3/4	5/6	7/8	9/10	11/12	13/14	15/16	17/18	19/20	Note
CN5	USB/ SD Card Extension	DA2	CDA	CK	DA0	0V	0V	DM	CNX	-	-	SD Card Pins 1-10
		DA3	3V3	0V	DA1	CD	5V	DP	0V	-	-	USB Pins 11-16
CN7	8x8 Keyboard Matrix and user I/O Ports	5V	3V3	K0	K2	K4	K6	K8	K10	K12	K14	3V3 output max 50mA
		0V	0V	K1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

Half duplex uses connector CN1 pins 1 and 8.

Pin Numbering - The square pad always indicates Pin1



Jumper and Additional Connector Information

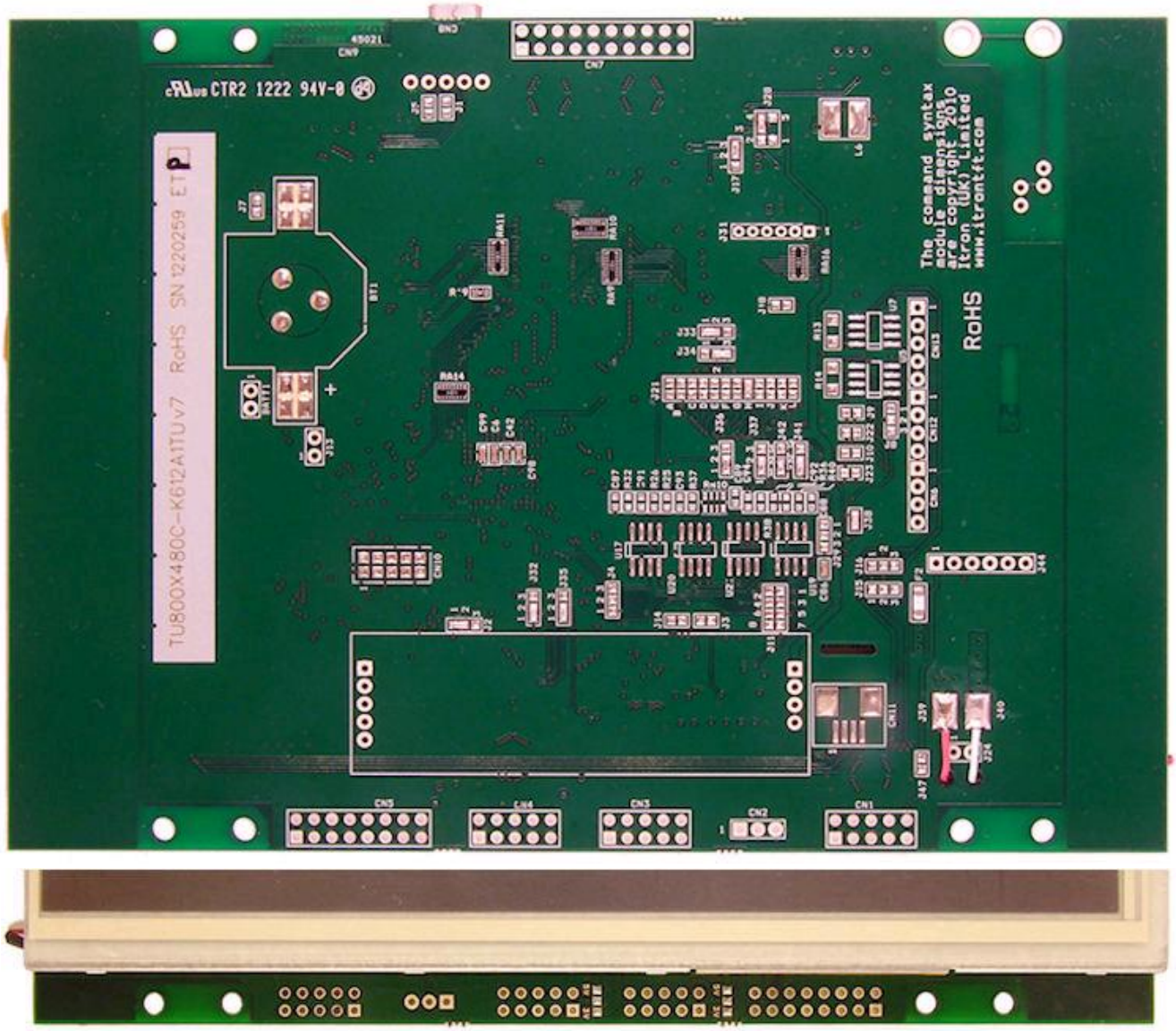
JMP/CON	Function	Note
BT1	Battery Connector	Apply solder bump to center pad before fitting holder. CR1216 battery positive up.
BATT1	RTC alternate power 3VDC	Apply right angle connector top side soldered.
BL	LED Backlight alternate supply	When the backlight is software disabled, 30VDC at 20mA can be applied by the user.
J4	RTS Jumper	Solder 1 and 2 for RTS.
J8	RS485 Half Duplex Jumper	Solder 1 and 2 for Full Duplex, Solder 2 and 3 for Half Duplex
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 TU800X480C-K612A1TUv2 with factory jumper setting.

Rear View TU800x480C-K611A1TUv3 with factory jumper setting
Front View TU800x480C-K611A1TUv3

Rear View TU800x480C-K612A1TUv7 with factory jumper setting



5V connect top 2 link
3V3 connect bottom 2 link

When using it as an output voltage this can be selected as 5V or 3V3
During reset the 3V3 is pulled to 0V in order to reset the external peripherals
The top right chassis mounting holes are connected to the TFT frame via J20 as default. Cut to isolate.
The TFT frame is connected to 0V via J19 as default. Cut to isolate.

Accessories - 2

Basket

Accessories

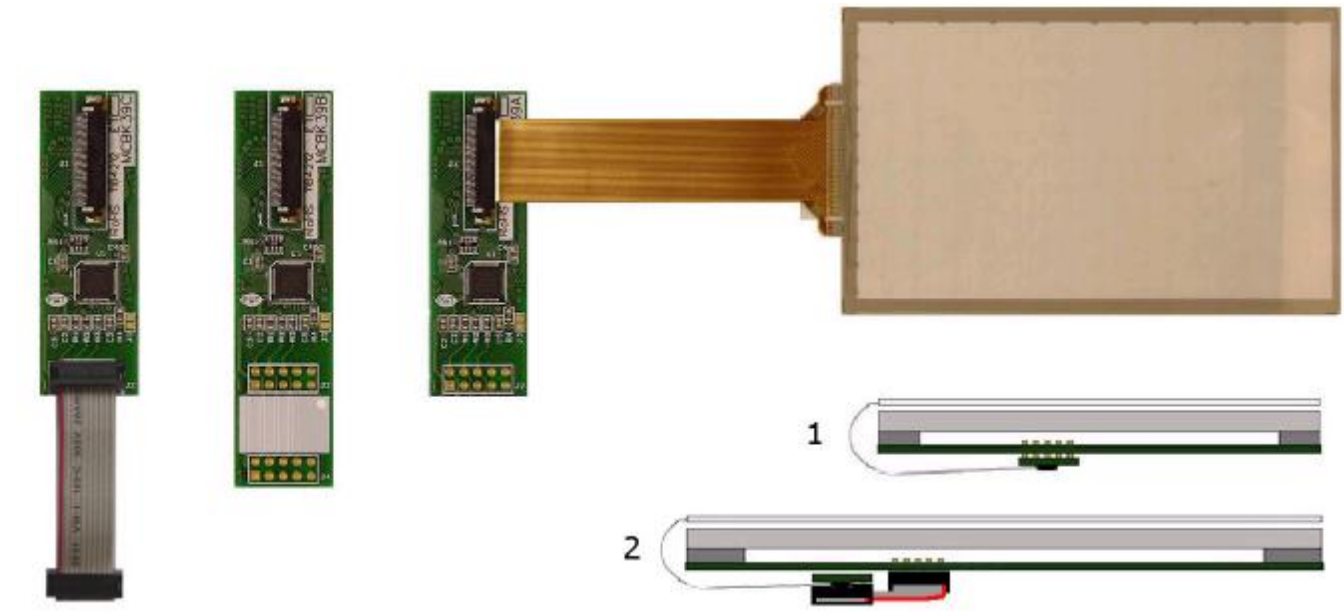
Noritake- Itron offers a range of accessories to get you up and running quickly.

Category	Part No	Description	Add to Basket	View
SD Card	micro SD Card 1GB	1GB micro SD card - Supplied with adaptor and demo software		
Capacitive Touch Panels	TPC043FGA	Capacitive Touch Panel for 4.3" TFT. Use -K612A1U suffix TFT modules		
	TPC057FGA	Capacitive Touch Panel for 5.7" TFT. Use -K612A1U suffix TFT modules		
	TPC070FGA	Capacitive Touch Panel for 7.0" TFT. Use -K612A1U suffix TFT modules		
Capacitive Touch Controllers	MCBK39A	Capacitive Touch Controller for 4.3" TFT		
	MCBK39B	Capacitive Touch Controller for 5.7" TFT		
	MCBK39C	Capacitive Touch Controller for 7.0" TFT.		
Audio Cards	MCBK-AC97P1	Audio Card with 2W stereo amp		
	MCBK-AC97P2	Audio Card with 0.5W mono speaker output.		
CAN Bus	EMBCK33A	CAN Bus Interface - Maximum speed 1MHz		
EMI Optical Filters	EFP105X067B07A	EMI Optical Filter for 4.3" TFT		
	EFP127X098B07A	EMI Optical Filter for 5.7" TFT		
	EFP164X104B07A	EMI Optical Filter for 7.0" TFT		
Rotary Encoder	MCB40A	Rotary Encoder with Push Switch		
Battery Holder	CONFSCR1216	Battery Holder - Uses a CR1216 battery - Solders to rear of TFT		
Cables	IFCK232-610A	RS232 Cable with 10 way IDC to 9 way female D type - 1000mm		
	IFCKUSBminiB2M	Type A to mini B USB cable - 2000mm		
	IFCK10IDC10-200A	10 way Ribbon Cable - 200mm		

Projective Capacitive Touch

PCT Adaptor Module connects to TFT CN3 and uses the I2C bus

Panel Part Numbers: 4.3"= TPC043FGA; 5.7"=TPC057FGA; 7.0"=TPC070FGA



MCBK39C (7.0") MCBK39B (5.7") MCBK39A (4.3")

1- How the MCBK39A/B connects

2- How the MCBK39C connects with ribbon cable (view mounting)

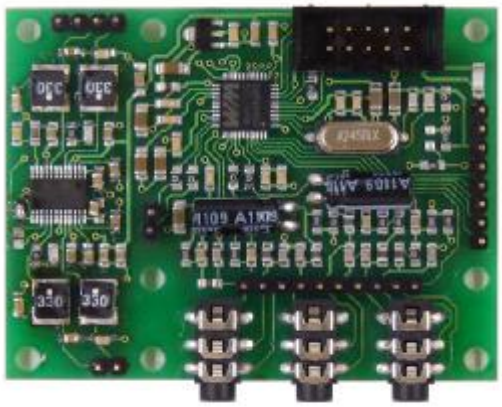
Capacitive Touch Setup and Example

NOTE: Can only be used where CN3 can be set to 3v3
4.3" PCB480272A Issue 8A or newer
5.7" & 7.0" PCB800480A Issue 3 or newer

AC97 Audio Module

MCBK-AC97P1
Bi-directional stereo codec and 8ohm mono speaker output with stereo power amp 2+2W

MCBK-AC97P2
Bi-directional stereo codec and 8ohm mono speaker output.
Minimum order 100pcs



2D mechanical

Datasheet -

2D mechanical

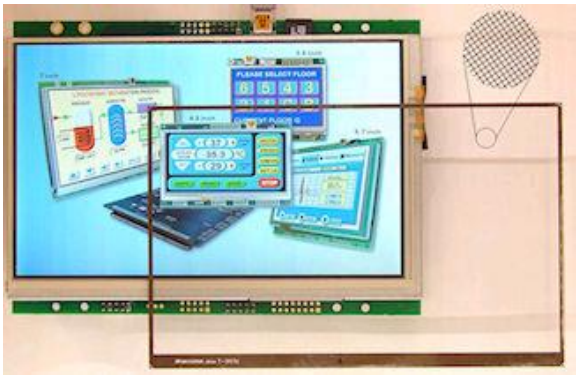
Datasheet -

CAN Bus Interface
EMBCK33A
Maximum speed 1MHz

More Details...

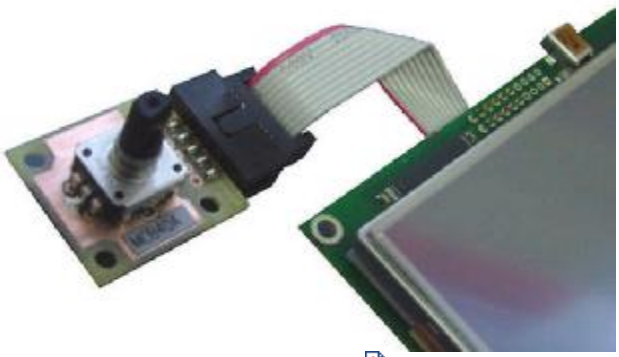
EMI Optical Filters

More Details...



1GB micro SD card
Supplied with adaptor

Rotary Encoder with Push Switch
MCB40A - Connects to CN7
2 rotary encoder modules work on a single 10 way IDC cable



2D mechanical
Datasheet

RS232 Cable
IFCK232-610A

USB Cable
IFCKUSBminiB2M

IDC Interface Cable
IFCK10IDC10-200A
10 way 200mm length

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

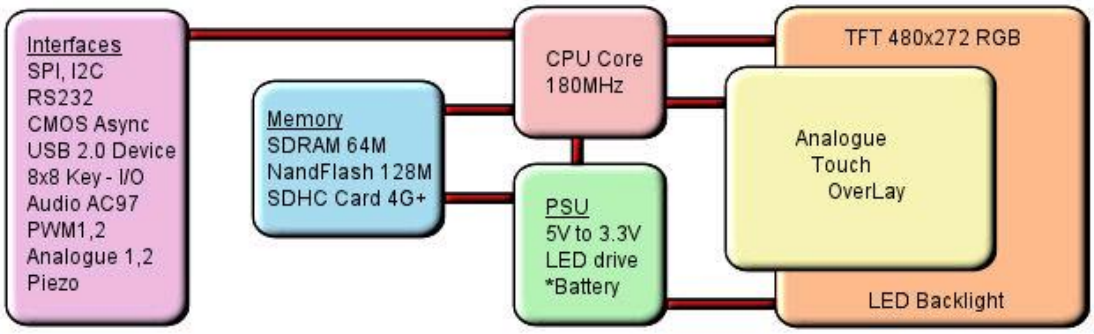
Overview - 3

itron SMART TFT Module Overview

Select Language

Product Overview
This product has been designed to simplify the implementation of TFT technology into your product yet provide a high level functionality. 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"

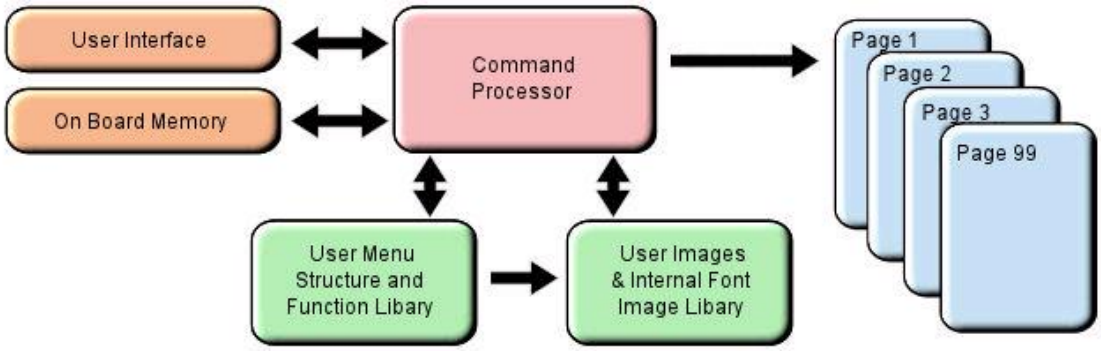


*option

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/ An existing host software requires only 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, format, 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
FPROG: Load Menu/Img to Flash	**PAGE:** Create a page of entities	**POSN:** Position cursor on page	**FUNC:** Create a function
LIB: Load Image/Font to RAM	**STYLE:** Set parameters	**TEXT:** draw text on page	**VAR:** Create a variable
INC: Include a sub file	**SHOW:** Show a page or entity	**DRAW:** draw box circle line graph	**IF ? :** Conditional test-true/false
RUN: Call function or user code	**HIDE:** Hide a page or entity	**IMG:** draw image on page	**LOOP:** Repeat commands
RESET: Reset system, NAND	**DEL:** Delete entity from Library	**KEY:**create touch or external key	**CALC:** Calculation and string edit
;; Refresh current page	**LOAD:** Copy and format pages, strings, interface and data		**WAIT:** Set delay period
; Terminate command			**INT:** 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 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;
```

Entities can be changed via the user interfaces by direct reference to there name or style with version v44 firmware.
Examples:

```
LOAD(homestyle.back,BLUE");          change the background colour of the page called homepage to blue
LOAD(rs2.set,"96e");                 change the rs232 baud rate to 9600 baud with even parity
LOAD(GenText.font,"40X56Kata");      change font size of all text using style GenText
```

We hope you find the 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

System Hardware Setup Parameters & Development Status - 4

System Hardware Setup Parameters and Development Status

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 November 2011.
The parameters for an interface are defined using the command setup(Name) {....}.

Parameters	Description	Status	View
asynchronous interfaces	set up rs2, rs4, as1, as2, dbg		RS2
set="96NC"	quick set up combination	OK	ASY
baud = num;	num = 110 to 6,250,000	OK	
data = num;	num = 5, 6, 7, 8	OK	
stop = num;	num = 1, 15, 2 15 is 1.5 bits	OK	
parity = ch;	parity = Odd, Even, None, Mark, Space	OK	
rxi= Y or C or N;	set receive buffer interface active	OK	
proc = ";" \\0D or other	process on receive string terminator	OK	
procDel = Y or N	delete or keep termination character.	OK	
rxb= num;	set size of receive buffer in bytes.	OK	
txi= Y or E or N;	set transmit buffer interface	OK	
txb= num;	set size of transmit buffer in bytes.	OK	
encode = s , w, m;	s= single byte ASCII, w=2 byte UNI, m= UTF8	OK	
flow = N , H, S;	flow control - none, hardware, software XON XOFF	OK	

spi interface	set up spi	Slave OK v43	SPI
set = "SHFC";	quick set up combination	OK	
active= M or S or N;	set as Master, Slave or None	Slave Only	
edge= LR, LF, HR or HF;	uses idle High or Low and Rising or Falling clock edge	OK	
speed = 100;	set transmit speed in master mode		
rxi= Y or C or N;	set receive buffer interface as active	OK	
proc=";" ,\\0D or other	process on receive string terminator	OK	
procDel = Y or N	delete or keep termination character.	OK	
encode = s , w, m;	s= single byte ASCII, w=2 byte UNI, m= UTF8	OK	
rxb= num;	set size of receive buffer in bytes	OK	
rxo= M or L;	set receive data order	OK	
rxf = N , H;	use none or hardware MB		
txi= Y or E or N;	set transmit buffer interface as active		
end= "\\nn";	byte returned when no data left in buffer		
dummy= "\\nn";	dummy byte sent to TFT to allow data to be sent to host		
txb= num;	set size of transmit buffer in bytes.		
txo= M or L;	set transmit data order		
txf = N , H;	none or hardware HB in Master mode		
txs = N , Y;	use select output \SS in Master mode		

i2c interface	set up i2c		I2C
set = "S7E";	quick set up of I2C - Slave and Address	OK	
addr= "nn"	address pair where nn write, nn+1 read	OK	
end= "nn"	byte returned when no data left in buffer	OK	
active= M or S or N;	set as Master Slave or None	OK	
speed = 100;	set transmit speed value in master mode	OK	
rxi= Y or C or N;	set receive buffer interface as active with command	OK	
proc = ";" \\0D or other	process on receive string terminator	OK	
procDel = Y or N	delete or keep termination character.	OK	
encode = s , w, m;	s= single byte ASCII, w=2 byte UNI, m= UTF8	OK	
rxb= num;	set size of receive buffer in bytes	OK	
txi= Y or E or N;	set transmit buffer interface as active with echo	OK	
txb= num;	set size of transmit buffer in bytes.	OK	

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

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

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

buzz = buzzer output	Use LOAD(BUZZ,var) var=ON,OFF, or time in millisecs	OK	BUZ

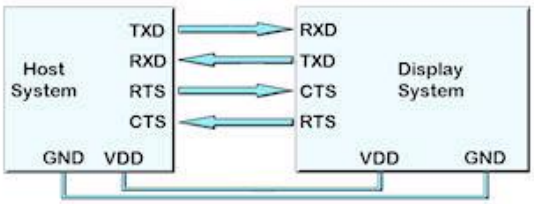
other interface references			
internal eeprom	parameter storage using extended variables VarE	OK	VAR

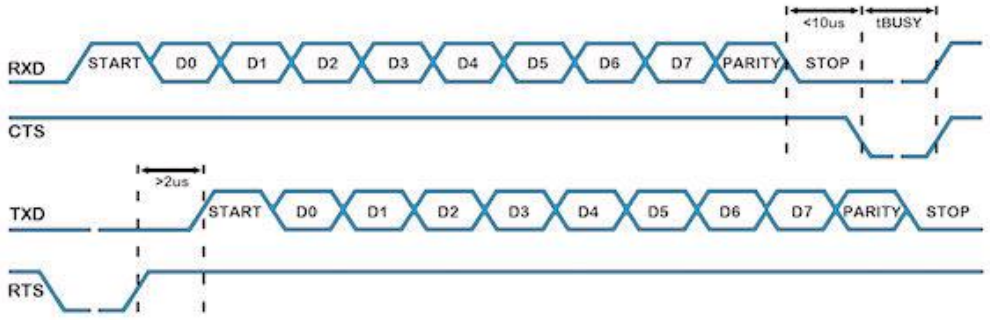
sdhc = SD Card (1G, 4G+)	FAT16/32 - 8 character file names, no directory. Not 2G	Read OK. Write v44	SD
internal NAND flash	Proprietary structure	Active v43 for firmware+ user code/images	NAND
usbcom = usb com port	use MCBK36A adaptor	v43	COM
CAN adaptor - 1MHz	adaptor connects to CN3	OK	CAN
ac97= audio buffer	adaptor connects to CN4	TBD	I2S

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
  procNum=5;         //interrupt on n bytes received as alternative to proc and procDel.
  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 text bytes and sd, wd and md for raw data.
  flow=N;            //none (N), hardware RTS/CTS or DTR/DSR (H), software XON XOFF (S).
  }
```

Data Processing Interrupt Characters

Termination characters can be specified to generate an interrupt to process a string of data.
The proc parameter is used in the port setup to define the termination characters.

```
proc = all;          <- trigger on all received characters
proc = CRLF;          <- trigger on a CR followed by LF (0Dh 0A)
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 (hex value)
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

Data Encode Modes

encode=s; 8 bit ASCII data. Codes 00-1F and 80-FF are converted to ASCII "\\00" - "\\1F", "\\
encode=sr; 8 bit data. Codes 00-07 are processed as cursor commands. 20-FF are processed as ASCII+ data
encode=sd; 8bit data. All bytes are processed as raw data.

Other mode styles are available:
D8M - 8 bit data with U16's, U32's etc output most significant byte first - same as sd
D8L - 8 bit data with U16's, U32's etc output least significant byte first
D16M - 16 bit data with bytes processed as most significant byte first - interrupt occurs after two bytes - same as wd
D16L - 16 bit data with bytes processed as least significant byte first - interrupt occurs after two bytes
D32M - 32 bit data with bytes processed as most significant byte first - interrupt occurs after four bytes - same as md
D32L - 32 bit data with bytes processed as least significant byte first - interrupt occurs after four bytes

Using hex pairs
sh or h8m or h8l = Ascii-Hex Char x 2 = U8; eg "A8" -> \\A8
h16m = Ascii-Hex Char x 4 = U16 (Most significant hex-pair first) eg "ABCD" -> \\ABCD
h16l = Ascii-Hex Char x 4 = U16 (Least significant hex-pair first) eg "ABCD" -> \\CDAB
h32m = Ascii-Hex Char x 8 = U32 (Most significant hex-pair first) eg "12345678" -> \\12345678
h32l = Ascii-Hex Char x 8 = U32 (Least significant hex-pair first) eg "12345678" -> \\78563412

Dot Operator

Parameters can be updated using the dot operator
LOAD(RS2.baud,19200);

LOAD(RS2.proc,"CR");

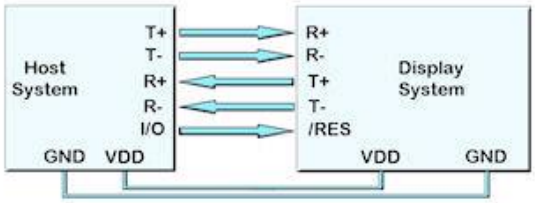
Example usage

```
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
  procNum=5;    //interrupt on n bytes received as alternative to proc and procDel.
  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 text bytes and sd, wd and md for raw data.
  flow=N;             //none (N), software XON XOFF (S).
  duplex=F;          //set Full Duplex (F) or Half Duplex (H).  Half duplex uses connector CN1 pins 1 and 8.
  }
```

Data Processing Interrupt Characters
Termination characters can be specified to generate an interrupt to process a string of data.
The proc parameter is used in the port setup to define the termination characters.

```
    proc = all;          <- trigger on all received characters
    proc = CRLF;          <- trigger on a CR followed by LF (0Dh 0A)
    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 (hex value)
    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

Data Encode Modes

```
    encode=s; 8 bit ASCII data. Codes 00-1F and 80-FF are converted to ASCII "\\00" - "\\1F", "\\
    encode=sr; 8 bit data. Codes 00-07 are processed as cursor commands. 20-FF are processed as ASCII+ data
    encode=sd; 8bit data. All bytes are processed as raw data.

    Other mode styles are available:
     D8M - 8 bit data with U16's, U32's etc output most significant byte first - same as sd
     D8L - 8 bit data with U16's, U32's etc output least significant byte first
     D16M - 16 bit data with bytes processed as most significant byte first - interrupt occurs after two bytes - same as wd
     D16L - 16 bit data with bytes processed as least significant byte first -  interrupt occurs after two bytes
     D32M - 32 bit data with bytes processed as most significant byte first -  interrupt occurs after four bytes - same as md
     D32L - 32 bit data with bytes processed as least significant byte first -  interrupt occurs after four bytes

     Using hex pairs
     sh or h8m or h8l = Ascii-Hex Char x 2 = U8; eg "A8" -> \\A8
     h16m = Ascii-Hex Char x 4 = U16 (Most significant hex-pair first) eg "ABCD" -> \\ABCD
     h16l = Ascii-Hex Char x 4 = U16 (Least significant hex-pair first) eg "ABCD" -> \\CDAB
     h32m = Ascii-Hex Char x 8 = U32 (Most significant hex-pair first) eg "12345678" -> \\12345678
     h32l = Ascii-Hex Char x 8 = U32 (Least significant hex-pair first) eg "12345678" -> \\78563412
```

Dot Operator
Parameters can be updated using the dot operator

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

Example usage

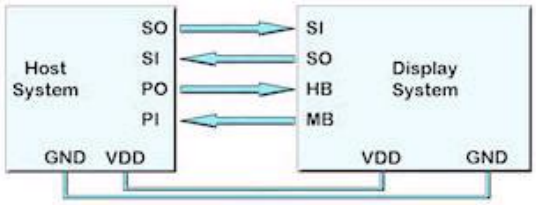
```
setup(RS4)
  {
  set="96NC"          //quick set up combination "48, 96, 192, 384, 768,  1150 with parity N, O, E and Command option".
  }
 PAGE( PageName, PageStyle)
  {
  POSN(100,100); TEXT ( RecvTxt, "Example", stRecvTxt);; //show received ASCII data on screen
   ....
   ....
   INT( SerRxInt,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

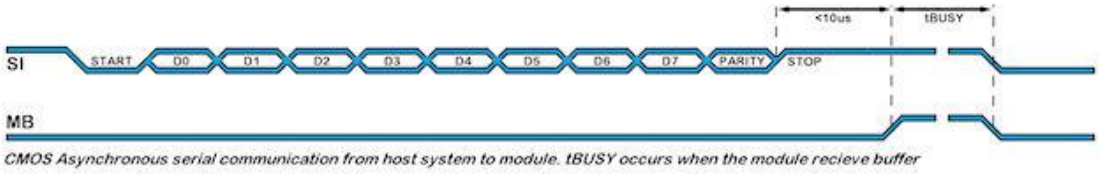
```
}

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

Operational

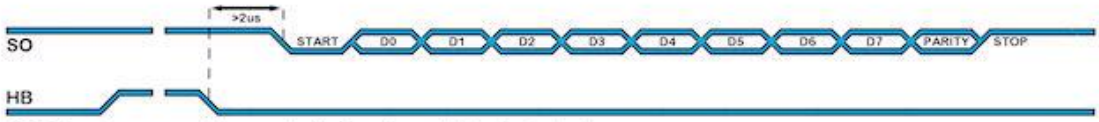
CMOS Asynchronous Interfaces - AS1, AS2, DBG (3v3 level)

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.





CMOS Asynchronous serial communication from host system to module. tBUSY occurs when the module recieve buffer



CMOS Asynchronous serial communication from the module to the host system.

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
  procNum=5;           //interrupt on n bytes received as alternative to proc and procDel.
  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 text bytes and sd, wd and md for raw data.
  flow=N;              //none (N), hardware RTS/CTS or DTR/DSR (H), software XON/XOFF (S).
  }
```

Data Processing Interrupt Characters

Termination characters can be specified to generate an interrupt to process a string of data.
The proc parameter is used in the port setup to define the termination characters.

```
proc = all;           <- trigger on all received characters
proc = CRLF;           <- trigger on a CR followed by LF (0Dh 0A)
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 (hex value)
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

Data Encode Modes

encode=s; 8 bit ASCII data. Codes 00-1F and 80-FF are converted to ASCII "\\00" - "\\1F", "\\

encode=sr; 8 bit data. Codes 00-07 are processed as cursor commands. 20-FF are processed as ASCII+ data

encode=sd; 8bit data. All bytes are processed as raw data.

Other mode styles are available:

D8M - 8 bit data with U16's, U32's etc output most significant byte first - same as sd
D8L - 8 bit data with U16's, U32's etc output least significant byte first
D16M - 16 bit data with bytes processed as most significant byte first - interrupt occurs after two bytes - same as wd
D16L - 16 bit data with bytes processed as least significant byte first - interrupt occurs after two bytes
D32M - 32 bit data with bytes processed as most significant byte first - interrupt occurs after four bytes - same as md
D32L - 32 bit data with bytes processed as least significant byte first - interrupt occurs after four bytes

Using hex pairs

sh or h8m or h8l = Ascii-Hex Char x 2 = U8; eg "A8" -> \\A8
h16m = Ascii-Hex Char x 4 = U16 (Most significant hex-pair first) eg "ABCD" -> \\ABCD
h16l = Ascii-Hex Char x 4 = U16 (Least significant hex-pair first) eg "ABCD" -> \\CDAB
h32m = Ascii-Hex Char x 8 = U32 (Most significant hex-pair first) eg "12345678" -> \\12345678
h32l = Ascii-Hex Char x 8 = U32 (Least significant hex-pair first) eg "12345678" -> \\78563412

Dot Operator
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.
   }
```

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, 15, 2  - note 15 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
   }
```

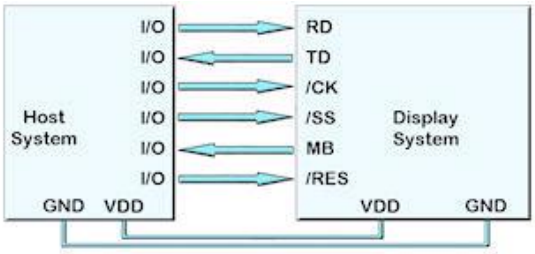
The default receive address for the adaptor is ID=155h with 11bit or 29bitID
packets accepted (2.0a or 2.0b spec)
All bytes are received on AS1 with 1 to 8 bytes of data.
The transmit ID is also 155H. with data sent via AS1 with data length of 1.

Connection to an iSMART TFT is shown below.

Active

SPI Interface

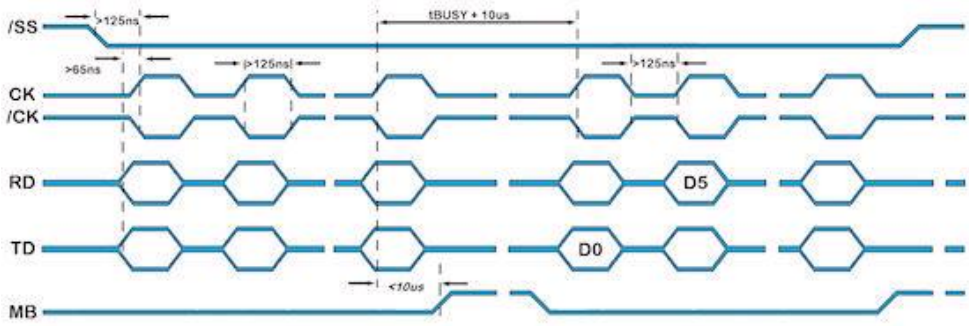
SPI Interface - SPI (3v3 level)
With synchronous communications enabled, data can be clocked into the TFT module using the rising or falling edge of SCK. This is selectable by the setup command which also sets other parameters. By default, data is clocked in on the rising edge with the most significant 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.



LINK the SPI jumpers on the back of the 4.3, 5.7 and 7.0 inch modules. Solder pads 1&2, 3&4, 5&6, 7&8.

In slave mode the /SS input must be pulled LOW if not used.
In slave mode the /IRQ pin is driven LOW to signify data is present in the transmit buffer.

Although the clock is capable of 90MHz, the practical speed is probably a maximum of 1MHz for external SPI communication. Please test your implementation extensively.



SPI Synchronous Serial Communication

spi - set up parameters

```
setup(spi)
  {
  set="SHRC";              //quick set up as (M)aster/(S)lave, idle (L)ow/(H)igh, edge (R)ising/(F)alling, (C)ommand and speed 350-90000
  }

setup(spi)
  {
  active=S;              //set as Master, Slave or None for both transmit and receive. Default = N
  mode=LR LF HF HR;    //set idle state Low or High and Rising or Falling clock edge. Default = LR
  speed=100;         //set transmit speed value in kilobits/sec from 350 to 90000 for master mode. Default = 1000
  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
  procNum=5;      //interrupt on n bytes received as alternative to proc and procDel.
  encode=s;        //set s=ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw text bytes and sd, wd and md for raw data.
  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
  txi=Y;               //set transmit buffer interface as active (Y), to echo command processing (E) or disable (N)
  end="\\nn";        //byte sent to host when no data left in display's spi transmit buffer.
  dummy="\\nn";  //dummy byte sent to module which is ignored so that data can be received by the host.
  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
  txs=N;             //use select output \SS in master mode. Default = N
  }
```

Data Processing Interrupt Characters
Termination characters can be specified to generate an interrupt to process a string of data.
The proc parameter is used in the port setup to define the termination characters.

```
proc = all;            <- trigger on all received characters
proc = CRLF;          <- trigger on a CR followed by LF (0Dh 0A)
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 (hex value)
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

Data Encode Modes
encode=s; 8 bit ASCII data. Codes 00-1F and 80-FF are converted to ASCII "\\00" - "\\1F", "\\

encode=sr; 8 bit data. Codes 00-07 are processed as cursor commands. 20-FF are processed as ASCII+ data
encode=sd; 8bit data. All bytes are processed as raw data.

Other mode styles are available:
D8M - 8 bit data with U16's, U32's etc output most significant byte first - same as sd
D8L - 8 bit data with U16's, U32's etc output least significant byte first
D16M - 16 bit data with bytes processed as most significant byte first - interrupt occurs after two bytes - same as wd
D16L - 16 bit data with bytes processed as least significant byte first - interrupt occurs after two bytes
D32M - 32 bit data with bytes processed as most significant byte first - interrupt occurs after four bytes - same as md
D32L - 32 bit data with bytes processed as least significant byte first - interrupt occurs after four bytes

Using hex pairs
sh or h8m or h8l = Ascii-Hex Char x 2 = U8; eg "A8" -> \\A8
h16m = Ascii-Hex Char x 4 = U16 (Most significant hex-pair first) eg "ABCD" -> \\ABCD
h16l = Ascii-Hex Char x 4 = U16 (Least significant hex-pair first) eg "ABCD" -> \\CDAB
h32m = Ascii-Hex Char x 8 = U32 (Most significant hex-pair first) eg "12345678" -> \\12345678
h32l = Ascii-Hex Char x 8 = U32 (Least significant hex-pair first) eg "12345678" -> \\78563412

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

Example usage

```
setup(SPI)
  {
  set="SHRY"          //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( SPIRxInt, SPIRXC, SPIRxEvent ); //Used when rxi=Y
   }

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

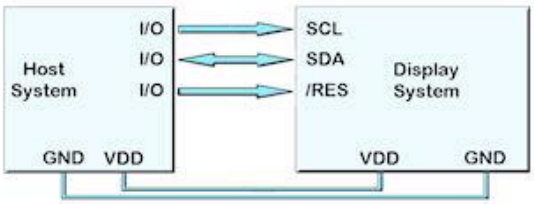
SPI slave active v47. Master planned for v48/49

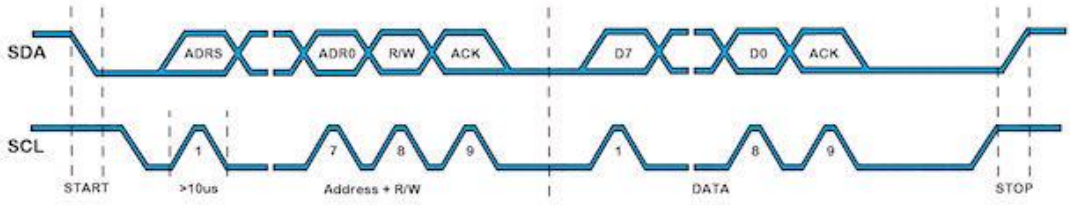
I2C Interface

TWI / I2C Interface - I2C (3v3 level)

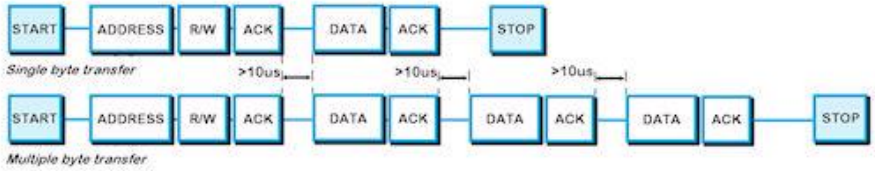
The I2C interface operates as a slave either in 'slave receive' or 'slave transmit' mode with a user defined address set in the I2C setup. Receive (i2c.rxb) and transmit (i2c.txb) buffers of 8192 bytes are created which can be cleared and set by the command processor. An end byte indicating empty buffer 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 9thclock 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
  procNum=5;     //interrupt on n bytes received as alternative to proc and procDel.
  encode=s;          //s= ASCII, w=UNICODE, m=UTF8 or use sr, wr and mr specifying raw text bytes and sd, wd and md for raw data.
  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=8192;          //set size of transmit buffer in bytes. Default = 8192 bytes
  }
```

When sending data in a protocol to the TFT module in slave mode, set up an interrupt either globally or in a PAGE for context functionality.
INT(I2Crecv,I2CRXC,I2Cfunc);

Data Processing Interrupt Characters

Termination characters can be specified to generate an interrupt to process a string of data.
The proc parameter is used in the port setup to define the termination characters.

```
proc = all;            <- trigger on all received characters
proc = CRLF;          <- trigger on a CR followed by LF (0Dh 0A)
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 (hex value)
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

Data Encode Modes

encode=s; 8 bit ASCII data. Codes 00-1F and 80-FF are converted to ASCII "\\00" - "\\1F", "\\
encode=sr; 8 bit data. Codes 00-07 are processed as cursor commands. 20-FF are processed as ASCII+ data
encode=sd; 8bit data. All bytes are processed as raw data.

Other mode styles are available:

D8M - 8 bit data with U16's, U32's etc output most significant byte first - same as sd
D8L - 8 bit data with U16's, U32's etc output least significant byte first
D16M - 16 bit data with bytes processed as most significant byte first - interrupt occurs after two bytes - same as wd
D16L - 16 bit data with bytes processed as least significant byte first - interrupt occurs after two bytes
D32M - 32 bit data with bytes processed as most significant byte first - interrupt occurs after four bytes - same as md
D32L - 32 bit data with bytes processed as least significant byte first - interrupt occurs after four bytes

Using hex pairs
sh or h8m or h8l = Ascii-Hex Char x 2 = U8; eg "A8" -> \\A8
h16m = Ascii-Hex Char x 4 = U16 (Most significant hex-pair first) eg "ABCD" -> \\ABCD
h16l = Ascii-Hex Char x 4 = U16 (Least significant hex-pair first) eg "ABCD" -> \\CDAB
h32m = Ascii-Hex Char x 8 = U32 (Most significant hex-pair first) eg "12345678" -> \\12345678
h32l = Ascii-Hex Char x 8 = U32 (Least significant hex-pair first) eg "12345678" -> \\78563412

Dot OperatorParameter 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 = \\FF;    //necessary to choose a character for end of string
speed = 100;
encode = sr;   //use raw data
rxi = Y;
txi = Y;
}

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

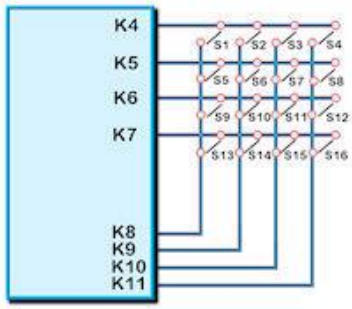
Keyboard and I/O Interfacing + PWM, ADC and Piezo

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

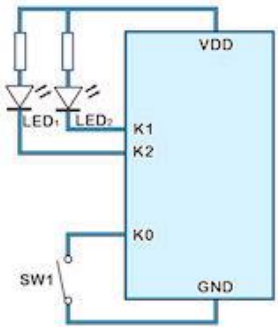


I/O Control
The module contains simple Input and Output functions for the 24 I/O lines (K0-K23).All inputs include an optional pull-up resistor ~50K-120K in value. The outputs can source ~1mA and sink ~3mA.
Certain I/O have expanded functions for customization.

At RESET or POWER ON, the I/O ports are initially pulled high.

K30 is the highest order bit and K0 the lowest.

NOTE: The ports K00 to K30 have series resistors and capacitors to GND.
Please check each model hardware specification for the specified values.
To use Kn connects to GND, use an INT(Name,Kn,function); command



keyio K00-K30 31 bits of user i/o and keyboard operatonal

```
setup(keyio)
 {
  active=\\000000FF;          //high is active "\\00000000" >"\\7FFFFFFF", default is inactive
  inp=\\0000000C;             //high is input, low is output "\\00000000" >"\\7FFFFFFF"
  trig=\\00000001;            //high is trigger interrupt "\\00000000" >"\\7FFFFFFF" as defined by edge. 1=trigger .
  edge=\\00000000;            //high is rising edge, low is falling edge "\\00000000" >"\\7FFFFFFF"
  keyb=\\00000FF0;            //high is scanned keyboard connection "\\00000000">"\\7FFFFFFF"
  pullup=\\7FFFFFFF;          //introduced in v49.06 to allow input pull up resistors to be set ON=1 and OFF = 0
  }
```

Single bit variables can be set and tested K00, K01, K02...K30 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

If another function is enabled which uses an I/O pin, the keyio functionality for the pin is disabled automatically.

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=\\00000001;        // set K00 to be active
  inp=\\00000001;           // set K00 as input
  trig=\\00000001;          // enable trigger interrupt on K00
  edge=\\00000000;          // 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-CNTK30

The 31 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 30.
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)
|	
CNT29	Counter on I/O K29 (CN3)
CNT30	Counter on I/O K30 (CN3)

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
```

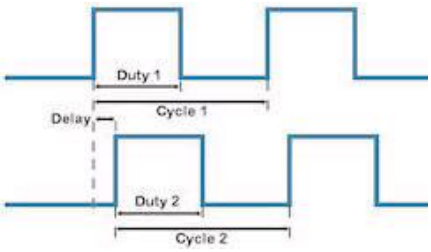
operational v40

pwm controller PWM1,PWM2,PWM3 **operational - PWM3 added v49.04**

```
setup(pwm)
 {
  active=123;                    //use 123 to synchronize PWM 1,2 and 3. N=none
     or
  active1=Y/N;   active2=Y/N;  active3=Y/N;

  pol1=H;                        //polarity = High or Low on first phase of PWM1
  pol2=H;                        //polarity = High or Low on first phase of PWM2
  pol3=H;
  cycle1="200";                  //cycle time in microseconds of PWM1. Range 160Hz to 1MHz
  cycle2 = "300";                //cycle time in microseconds of PWM2. Range 160Hz to 1MHz
  cycle3 = "300";                //cycle time in microseconds of PWM3. 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
  duty3= "56";                   //value of first phase as a  percentage for PWM3 = 1-99
  delay= "50";                   //delay between first phase of PWM1 and first phase of PWM2 in microseconds
  }
```

PWM1 is pin 5 on CN4 and PWM2 is pin 6 on CN4. PWM3 is available on CN7 pin 14.
Duty cycle values <1 are forced to 1 and values >99 are forced to 99 to prevent a DC condition.
The parameter values can be adjusted using the LOAD command with a dot (.) operator....LOAD(pwm.cycle1,350);

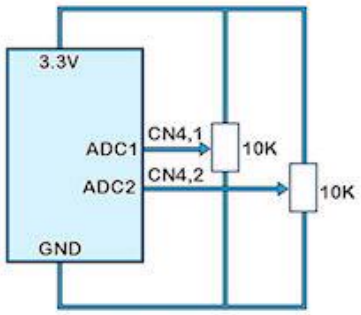
or using an adjustable variable as in LOAD(pwm.cycle1,cycvar);



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
  }
```



Input impedance 150R - 6500R

example usage

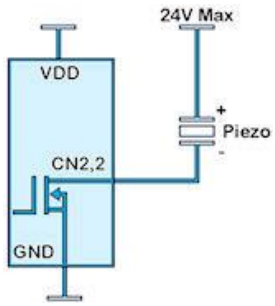
```
//TU480A.MNU Menu file for TU480X272C with single red pen.
STYLE(BlackPg, Page) { Back=\\00FF66;} //green background
STYLE(stGraphRed,DRAW){type=graph; col=red; width=4; maxX=490; maxY=300; curRel=CC; }  //red pen for graph
SETUP( adc ){active=1; calib1=0.2; avg1=8; }

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.

---

rotary encoder control

2 encoders to be connected to any available I/O ports.
The rotary encoders available on the Accessories page allow a 10way IDC cable to control 2 encoders with center push switches.

```
SETUP(ENC)
{
active = 1/2/12; // N=none active, 1=enc1 active, 2 = enc2 active, 12 = both active
a1 = \\xx; // port number for enc1 A channel
b1 = \\xx; // port number for enc1 B channel
a2 = \\xx; // port number for enc2 A channel
b2 = \\xx; // port number for enc2 B channel
debounce1 = n; // debounce time in ms for enc1 (1 - 100ms)
debounce2 = n; // debounce time in ms for enc2 (1 - 100ms)
timeout1 = n; // timeout period in ms for enc1 (1 - 1000ms)
timeout2 = n; // timeout period in ms for enc2 (1 - 1000ms)
mode1 = n; // encoder type (1 or 2) for enc1
mode2 = n; // encoder type (1 or 2) for enc2
}

INT(name, ENC1, fnc);
INT(name, ENC2, fnc);
```

* System variables ENCVAL1 or ENCVAL2 are type S32 and hold the current count values for each encoder.
* LOAD(ENCVAL1,n); presets encoder variable value to required number n.
* The value of ENCVALn is decremented by the number of clicks left and incremented by the number of clicks right.
* Timeout specifies the time from the last rotation event until the INT is triggered.

SD Nand EEprom and USB Interface - 11

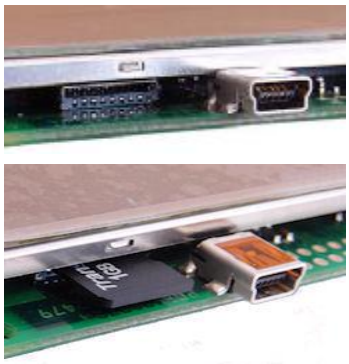
SD Card

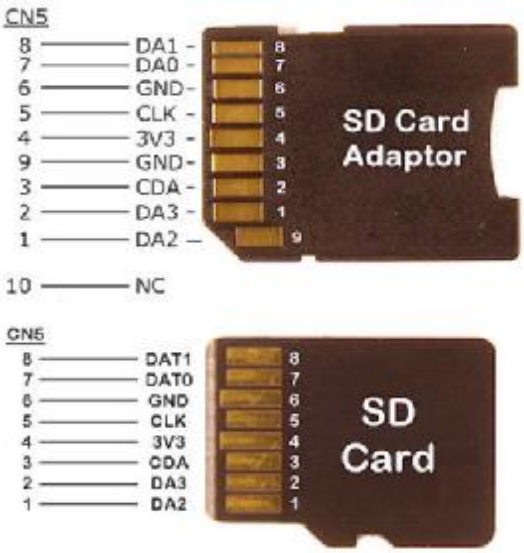
The SD card slot supports 1G SD in FAT16 or FAT 32 format and 4G, 8G and 16G SDHC cards in FAT32 format.
When loading new boot file updates, a 1G SD card must be used.
8.3 character file names are supported.

An SD card can be used to load production software into the internal NAND flash memory using the FPROG and LOAD command. An option to encrypt this will be available.

Large files can be retained on SD card for access during operation of the application for slide shows and tutorials.

Data can be written to the SD card for data logging purposes.
(planned operational v48/v49)





The SD Card can be connected to CN5 on the TFT using a suitable cable and adaptor.

The pictures to the left show the pin out of the SD Card Adaptor and SD Card with respect to the pins on CN5

The picture on the right shows an 800mm cable with various connectors and cable stubs to evaluate SD card reading which proved successful when uploading the 88 files of the demonstration software.

We recommend the use of screened flat IDC cable of minimum length with a ferrite collar.

LONG CABLE TEST

SD Card Handler v49.02+
Unlimited sub directory support with long file names.
> LIB(libImg, "SDHC/dir1/dir2/dir3/dir4/file.ext");
> CALC(txtVar, "SDHC/dir1/dir2/dir3/dir4/", "DIR");

Write to / Append file on SD card added (if file does not exist, it is created)
> LOAD("SDHC/file.ext", var, var, ...);
> LOAD("SDHC/dir1/dir2/dir3/dir4/file.ext", var, var, ...);

Read text data from the SD card
STYLE(stBuff,data) {type=TEXT;length=100000;} //define buffer
VAR(databuffer,"",stBuff); //create buffer to hold data
CALC(databuffer,"SDHC/file.ext,"FREAD"); //read file into buffer
Use a CALC(SPLIT) command to extract the data from the buffer as required.

> Limitations:
Filename has to be specified in quotes, variable for filename not currently supported.
Files are written in 'SR' encoding, ie raw ASCII.

RESET(SDHC); which reinitialises SD card handler (useful after SD card removal/reinsertion).
Improvements made. 33% clock speed increase. Code simplified.

NAND Flash

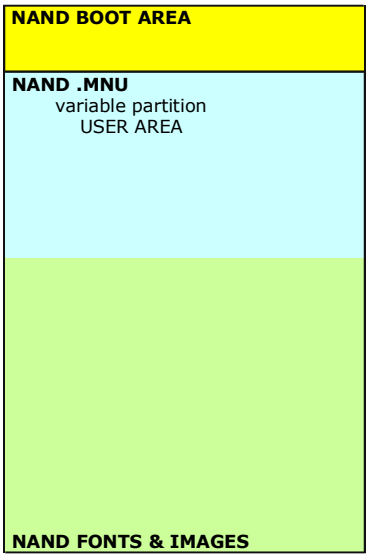
The 128Mbyte NAND Flash memory is organised into 3 drives. A protected drive containing boot and operating files ~ 4Mbyte, a user accessible menu file drive and an image / font file drive with variable partition to allow large image and fonts to occupy the maximum space of 124Mbyte

The user area of the NAND flash can be cleared using the RESET(NAND) command.

Data can be written to the NAND flash from an SD card or via the serial interfaces using the FPROG and LOAD commands

Only file names of mnu, fnt and image files can be sent to a serial port to protect user IPR. Encryption will be available.

LOAD(NAND, "file"); loads "file" into MNU area of NAND if it is a mnu file else into LIB area of NAND for other files
LOAD(NANDLIB, "file"); loads the "file" into LIB area of NAND
LOAD(NANDMNU, "file"); loads the "file" into MNU area of NAND
RESET(NAND); clears both MNU and LIB areas of NAND
RESET(NANDLIB); clears just the LIB area of NAND
RESET(NANDMNU); clears just the MNU area of NAND
LIB(imgnam1,"NAND/filename.bmp...."); Only use the name NAND to read files
Note, when reading a file from NAND, both areas of NAND are looked at automatically.
Care should be taken not to put the same named file into both areas. The LIB area is read first.



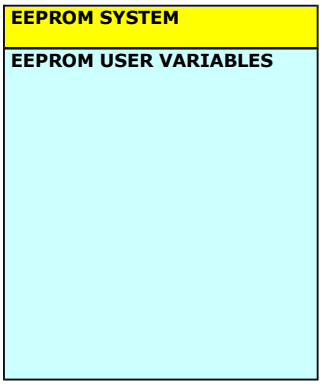
EEPROM

The internal EEPROM has 7.5K bytes of user space and 500 bytes for system parameters like touch screen calibration and screen orientation. Data variables can be created for storage in EEPROM with the VAR command. These are protected by checksum and in the event of corruption, the default value assigned to a variable will be used.

It may be necessary to clear the EEPROM with a RESET(EEPROM); command. After this default parameters will be applied for touch screen calibration and orientation.

The process of installing a new boot file (boot.bin) will also clear the EEPROM.

Operational



Internal USB Device

The connector CN8 allows users of USB enabled modules to connect directly to a PC using a standard miniB cable.
Files can then be upload to the itron SMART TFT internal NAND or SDRAM using a terminal software or the iDEVTFT development environment.

To enable the USB when a TUxxx.mnu file is operational include the following program code:

SETUP(USB) {rxi=C;txi=Y;rxb=250000;}

When the TFT modue is connected to a PC, a pop up will indicate a new device has been detected. Download the USB .INF files applicable to the PC OS and install as directed.

If a TUxxx.mnu file is not present in the NAND or SD card, the module will enable USB communication and installation can be achieved using the appropriate .INF file.

USB Enabled Hardware

Module	K611	K612
320X240	v2+	v3+
480X272	v4+	v7+
640X480	v3+	v3+
800X480	v3+	v3+

Download Internal USB Driver
Windows XP/7

There are 2 .INF files in TUxxUSB1.ZIP
Un ZIP and locate in an accessible directory on the PC and use the .INF for your PC operating sytem.

You may need administrator rights to install the USB.

USB Interface Adaptor MCBK36A/B

An adaptor MCBK36A is available to provide a USB interface to a PC via the async port AS1 on CN3. The baud rate is pre-set at 500kHz on the MCBK36A.

Setup your AS1 port in your TUxxx.mnu file with SETUP(AS1) {baud=500000;rxi=C;txi=Y;flow=H;} or download this file and copy to an SD card.
You may need to rename it to the start up file required by your size of module (TU320A,TU640A,TU800A).
This is also useful in case the AS1 setup gets changed in NAND during development when used by other peripherals.

The MCBK36A can be directly soldered into the TFT module with the square solder pad (pin 1) on each PCB common or connected via a ribbon cable. The 5V to power the module can also be derived from the PC.

During first plug in of the USB adaptor, your PC will ask for a USB driver which can be downloaded here. This .inf file requests the PC to use the standard com port driver supplied with Windows.

Download XP Driver Here

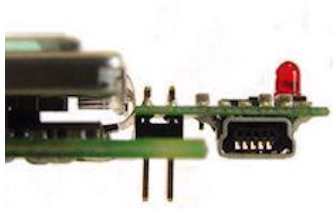
Download Win7 Driver Here

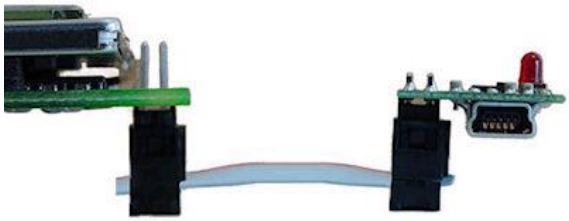
The iDevTFT software can upload files to the TFT module during the development phase as well as send test commands to emulate slave operation.

The best strategy is to load your images and fonts to NAND flash via SD card, then update your .mnu file changes via USB to the RAM. Once complete, These can then be transferred to NAND.

Note that when SD card and NAND are empty, AS1 and RS2 are automatically initialised to receive command data.
However, once NAND has a .mnu file installed, this must include the SETUP(portname) command otherwise communication with the AS1/RS2 ports will not be possible.

Datasheet MCBK36A Operational v45





Connection via CN3

You can force the iSMART AS1 port to default communication with the MCBK36A operation by linking connector CN6 pins 3 & 4 on the iSMART module.

Command Overview - 12

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
FPROG.....FEND	Store menu and image files in onboard flash	OK	FPROG
LIB(Name,Source)	Load picture,audio or font into library. BMP/JPG/WAV/FNT	OK	LIB
INC(FileName)	Include the contents of another menu, style or setup file	OK	INC
RUN(Func)	Run a function or user code	OK except custom code	RUN
RESET(Name)	Clear eeprom, delete list, library, or reset system	OK except deleted	RESET
LOAD(Name,N2,N3,N..)	Multi function copy pages, variable N2--N.. to Name.	OK	LOAD
SHOW(Name)	Show a page, entity	OK	SHOW
HIDE(Name)	Hide a page, entity	OK	HIDE
DEL(Name)	Delete a page, entity	OK	DEL
VAR(Name,Value,Style)	Create a variable of a specified type with a default value	OK	VAR
VAR(Name,Value,Style,Num)	Create an array of variables with size num	Plan	Plan
STRUCT(Name,Num) { }	Create a multi-dimensional structure of arrays or variables	Plan	Plan
IF(Var~Var?Func1:Func2)	Evaluate condition and do func1 if true, func2 if false	OK	IF
SELECT(var) { CASE(n,func);}	Evaluate a variable and undertake function	Plan	Plan
LOOP(Name,Var){...}	Loop for a specified number of times	OK	LOOP
INT(Name,Buffer,Function)	If interrupt triggered do function	OK	INT
CALC(Result, Var1, Var2, Act)	Quick calculation and text manipulation	OK	CALC
FUNC(Name) {...}	Declare a set of commands	OK	FUNC
STYLE(Name,Type) {...}	Predefine parameters for page entities and variables	OK	
WAIT(Time)	Wait specified milliseconds before next	OK	WAIT
;	Terminate command	OK	SEMI
;;	Refresh current page	OK	DSEMI
[cmd();cmd();....cmd;]	Enclose commands as inline function in IF, INT, KEY, RUN	OK	INLINE
POSN(X,Y,Page/Name,Style)	Position cursor or re-position named entity	OK	POSN
PAGE(Name,Style) {..}	Specify contents of page		PAGE
sizeX, sixeY	Specify the size of the page	OK except large size	-
posX, posY	Specify the absolute position on screen	OK	-
back	Specify the background colour of page	OK	-
image	Specify a background image for the page	OK	-
TEXT(Name,Text,Style)	Define text		TEXT
font	The ASCII based + extended fonts	OK	-
size	Size multiplier ie 2 = 24x24 to 48x48	OK	-
col	Specify the text color.	OK	-
maxLen	Specify the maximum number per row (Max 512)	OK	-
maxRows	Specify the maximum number of rows (Max 32)	OK	-
curRel	Specify the relative placement of the text	OK	-
rotate	Specify the rotation of the text 0,90,180,270	OK	-
DRAW(Name,X,Y,Style)	Create box, circle, line, pixel, trace graph.		DRAW
type	Specify the type of shape to draw	OK	-
col	Specify the border colour of the shape	OK	-
back	Specify the back colour of the shape	OK	-
width	Specify the border width of the shape	OK	-
sizeX,sizeY	Specify the maximum width and height	OK	-
curRel	Specify the relative placement	OK	-
rotate	Specify the rotation of the shape 0,90,180,270	OK	-
IMG(Name,Source,Style)	Image placement and manipulation		IMG
scale	The image can be scaled 50%,75%,200%.300%	OK	-
sizeX, sizeY	Specify the maximum width and height	OK	-
curRel	Specify the relative placement .	OK	-
rotate	Specify the rotation of the image 0,90,180,270	OK	-
action	Moving and image onto the screen for slide shows	OK	-
step	Number of pixels moved in a sliding image	OK	-
KEY	Designation of touch or external key matrix		KEY
type	Specify the source of key data - touch or external	OK	-
debounce	Specify the time delay to allow a key	OK	-
delay	Specify the time delay for auto repeat	OK	-
repeat	Specify the time delay for auto repeat	OK	-
action	Specify action point as Down or Up	OK	-
curRel	Specify the relative placement	OK	-
System Setup	set up main display system		SYS
bled	set LED backlight 0=OFF, 100=full ON or 1-99	OK	
wdog	set watchdog time to OFF, 100ms, 500ms or 1 second	OK	
encode	s= single byte ASCII, w=2 byte UNI, m= UTF8	OK	
test	show or hide outline view of touch areas on screen.	OK	
calibrate	calibrate the touch screen	OK	
touchenable	enable=y or disable=n touch keys	OK	

touchsamples	set the number of samples per touch test	OK	
touchdebounce	set the time period between touch tests	OK	
touchaccuracy	set the matching accuracy between consecutive tests	OK	
angles	set the system angles ot degrees or radians	OK	
startup	set screen start up messages to none, bar or all.	OK	
rotate	set screen orientation with respect to PCB.	0,180. Plan 90,270	
clkfreq	set the external bus clkfreq. 80MHz to 92MHz	OK	

Real Time Clock and Date	Specify real time clock	OK	RTC
RTCSECS	numeric variable containing seconds (0-59)		
RTCMINS	numeric variable containing minutes (0-59)		
RTCHOURS	numeric variable containing hours (0-23)		
RTCDAYS	numeric variable containing days (1-31)		
RTCMONTHS	numeric variable containing month (1-12)		
RTCYEARS	numeric variable containing year (1900-2099)		

Real Time Clock Alarm	Specify real time clock alarm	OK	RTC
RTASECS	numeric variable containing seconds (0-59)		
RTAMINS	numeric variable containing minutes (0-59)		
RTAHOURS	numeric variable containing hours (0-23)		
RTADAYS	numeric variable containing days (1-31)		
RTAMONTHS	numeric variable containing month (1-12)		
RTAYEARS	numeric variable containing year (1900-2099)		

Run Time Counter	Predefined variables which can be set and tested.	OK	
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 day 0-n		
CNTRUN	millisecond increments from last system reset	OK	

I/O Counters	The 24 I/O can have a software counter connected to each	OK	
CNTK00 - CNTK23	Counter connected to K0 through to K23		

System Setup

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;	display messages and progress bar at start up 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; or 180;	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
angles=degrees;	select degrees or radians for calc functions
encode = s, w, m;	menu text strings can contain single byte ASCII (s), 2 bytes for UNIcode (w) or multibyte for UTF8 (m)
calibrate = y;	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};
clkfreq=92000000;	Main external bus clock is changeable in 2MHz steps from 80MHz to 92MHz (default).
ignore=allErrors;	Ignores all errors and continues execution (only recommended in test as can cause undesired results). =invalidJpg; Ignores errors for unsupported JPG formats (eg progressive) and the image is skipped =imageTooBig; Ignores errors when there's not enough memory to load image and the image is skipped

Example system set up

```
setup(system)
  {
  bled=100;
  wdog=100;
  rotate=0;
  calibrate=n;
  test=showTouchAreas;
  angles=degrees;
  startup=all;
  encode=s;
  clkfreq=92000000;
  }
```

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

RESET(Name)

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 NAND flash memory with RESET(NAND);
Clear the MNU files in NAND flash with RESET(NANDMNU);
Clear the BMP, FNT, WAV files in NAND flash with RESET(NANDLIB);
Clear the library with RESET(LIBRARY); //Allows new program to load. Interface setup unchanged.

Note: When a RESET(SYSTEM); or hardware reset occurs, the boot software in the module looks to see if a valid start up file type TUxxxA.mnu is present on the SD card, if not, it looks in the internal NAND flash memory. If no TUxxx.mnu file is found, the module initializes the interface RS2 in command mode as 115200,8,1,N and AS1 as 500000,8,N,1,H.

Reset 'Deleted' TBA

FPROG ..... FEND

FPROG and FEND are used to program subsequent commands into internal flash memory. Use the RESET(NAND) command after FPROG if the existing files are to be replaced, otherwise the files are appended to NAND. Subsequent LIB commands then load images and files from NAND. When the module starts up, it checks for the correct TUxx.mnu start file in NAND, loads it into memory but skips the code between FPROG and FEND

Example content of TU480A.mnu file on SDHC card

```
FPROG;
RESET(NAND);
LOAD(NAND,"SDHC/TU480A.mnu"); //copies itself
LOAD(NAND,”SDHC/imgfile1.bmp”);
LOAD(NAND,”SDHC/imgfile2.bmp”);
FEND;


LIB(img1,”NAND/imgfile1.bmp”);
LIB(img2,”NAND/imgfile2.bmp”);
LIB(img3,"SDHC/img3.bmp");   //loaded from SDHC each boot up.
etc
```

Each file is copied into a buffer before writing to NAND.

A file called FPROG.MNU can be created by accessing the FILES >> Create Project Flash File in the iDevTFT development software. This can be included into the main TUxxx.mnu file using the INC command as follows:

```
FPROG
INC("SDHC/fprog.mnu");
FEND
```

v43.

INC(Source)

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.

* Maximum line length increased from 512 bytes to 8K bytes for include files.

Example: INC("sdhc/submenu.mnu") specifies the file path on the SDcard.
INC(File1,File2,File3,...FileN); multiple files are possible
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:

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.

RTCSECS	numeric variable containing seconds (0-59) which can be tested or loaded into a text.
RTCMINS	numeric variable containing minutes (0-59) which can be tested or loaded into a text.
RTCHOURS	numeric variable containing hours (0-23) which can be tested or loaded into a text.
RTCDAYS	numeric variable containing days (1-31) which can be tested or loaded into a text.
RTCMONTHS	numeric variable containing month (1-12) which can be tested or loaded into a text.
RTCYEARS	numeric variable containing year (1900-2099) which can be tested or loaded into a text.

RTC Day Of Week

Added day of week support to RTC.
Built in variable RTCWEEKDAY reports day of week where 1=Monday, 2=Tuesday,... 7=Sunday.
Formatting parameters added for RTC
D Short textual representation of day, three letters: Mon-Sun
L Full textual representation of the day of the week: Monday-Sunday
N ISO-8601 numeric representation of the day of the week: 1 (for Monday) - 7 (for Sunday)

Note RTC day of week is indeterminate if RTC has not been set.
The RTC Alarm does not support day of week.
For an alarm that triggers every Thursday at 16:00, the following example can be used:

```
INT( RTA, fnc_Alarm );
LOAD( RTA, ":::16:00:00" );
FUNC( fnc_Alarm )
  {
   IF( RTCWEEKDAY != 4 ? [ EXIT( fnc_Alarm ); ] );
     // Do Thursday alarm code here...
  }
```

Real Time Clock Alarm (RTA)

Support for an RTC Alarm is provided using RTA. This can be set for duration, time or time and date.
You can set an alarm every 20 seconds, at 17.45 every day or on the 15th March at 12.52 each year.
To setup the interrupt which is triggered at the alarm point:

```
INT(name,RTA,function);
```

To load the alarm time, use same format as setting RTC.
Only populated values are used to set the alarm, therefore alarms can be set to go off every minute, hour, hour:minute:second, day or month etc..
Note, the alarm does not support the years parameter, and is ignored when setting the alarm.

Setting the alarm:

```
LOAD(RTA,":5:26:14:7:03"); // Alarm will occur every year on 26th May at 14:07:03
LOAD(RTA,":::13:15:");   // Alarm will occur every day at 13:15:00
LOAD(RTA,":::",hours,":",mins,":",secs); // Alarm will occur every day at hours:mins:secs
LOAD(RTA,":::::20"); // Alarm will occur every 20 seconds past the minute.
```

To clear alarm:

```
LOAD(RTA,0); // Clear Alarm
LOAD(RTA,":::::"); // Clear Alarm
```

Settings can be read by accessing the built in variables
RTAYEARS, RTAMONTHS, RTADAYS, RTAHOURS, RTAMINS, RTASECS
If a value has not been set then -1 is returned.

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)
|	
CNTK22	Counter on I/O K22 (CN4)
CNTK23	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

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
CNTRUN	Increments every millisecond since system reset. 86,400,000 = 1 day.

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
```

RunTime Counter Interrupts

Wrap-around interrupt for the RunTime counter have been added.

```
INT(name,CNTMILLI,function);  // function called every 1000ms
INT(name,CNTSECS,function);   // function called every 60s
INT(name,CNTMINS,function);   // function called every 60mins
INT(name,CNTHOURS,function);  // function called every 24hours
INT(name,CNTDAYS,function);   // function called every 2^32days
```

For timer resolutions of less than a second, use TIMER0 to TIMER9.

operational

Timers (TIMER0 - TIMER9)

Ten (10) count-down timers with 1ms resolution - TIMER0 to TIMER9

To setup the interrupts:
INT(name,TIMER0,function); to INT(name,TIMER9,function);

To read the remain time before expiry
LOAD(var,TIMER0);

To run the timer once
LOAD(TIMER0,time); // time is in milliseconds

To run the timer multiple times
LOAD(TIMER0,time,num); // time is in milliseconds
num is number of times timer runs, 1 = 1 time, 2 = 2 times etc, 0 = non-stop

To clear the timer
LOAD(TIMER0,0);

Example Timer Usage

```
LOAD(TIMER6,1000);   // TIMER6 runs once and expires after one second
LOAD(TIMER9,1000,0); // TIMER9 runs forever, expiring every second
LOAD(TIMER4,500,5);  // TIMER4 runs five times, expiring every 500ms
LOAD(TIMER3,0);      // Clear TIMER3
LOAD(TIMER7,time);   // TIMER7 runs once and expires after value in var time
```

operational

WAIT(Time)

Wait for a period of milliseconds before processing menu commands.
Wait timer accuracy of 1ms ±200ns.
Interrupts and key presses still occur during the wait period and can be processed.
Restriction: If the WAIT() command is within a function called from a KEY() command then further key presses will be ignored. Each touch key press function must be processed to completion before another can be processed. Please refer to the project example 'keyboard' for the technique to process keys.

Operational.

PAGE(Name,Style) {.....}

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.

Example:
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
 }
```

Page Refreshing v47
A new mode has been added for pages which allows for either
All entities to be redrawn on a page when a double semicolon (;;) refresh is encountered (default),
or
Only the entities that have been modified since last refresh to be redrawn when a double semicolon (;;) is encountered.
A SHOW(page); command will always redraw all the entities on a page.

```
STYLE(name,PAGE) { update=all; } // default: refreshes all entities on a ;;
STYLE(name,PAGE) { update=changed; } // refreshes only changed entities on a ;;
```

Page Styles
The style defines the page size, position and background.

```
STYLE(stPage,Page)   //create a style name and define as type Page
 {
 update=all; //define page refresh method with ;; options: 'all' or 'changed'
 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 for the page using the entity name used in the LIB command to store the image .
 }
```

Using update=changed
To gain the faster refreshing, a few rules apply.
1/ Only the screen area where the changed entity is located is redrawn.
2/ The entity is redrawn on top of any existing pixels being displayed in that area.
3/ Entities with transparent backgrounds will show all previous rendering at that location in the transparent area.
4/ Hiding an entity will not produce any visible difference until a full page refresh is performed.

To support the “update=changed;” method.
1/ Do not use images with transparent backgrounds
2/ Specify the “back” colour in the style for text.
3/ To hide an entity, a “masking” image will need to be placed over the entity.
4/ To refresh only the entities on a page that have been modified, use the double semi-colon “;;” refresh method after the last update object, e.g. IMG(imgt1,myimg);;.
5/ To refresh the whole page, use the SHOW(page); method.

POSN(X,Y,Name/Page)

Position Cursor X,Y or +X or –X or X, Y, Name/Page.
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.

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.

It may be necessary to place the cursor on another page to create a new entity.

Examples:
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.

Move multiple entities at the same time
Multiple entities can be moved at the same time POSN(x0,y0,Ent1,Ent2,Ent3,Ent4,...);
POSN(+10,+10,Img1,keyup,keydn); moves both image and keys 10 pixels in X and Y direction
This is useful for slider bars where the bar image, key up and key down objects move in sync.

operational

TEXT(Name,Text,Style)

Create or update Text.

Use Carriage Return and/or Line Feed for multi line entry "\\0D\\0A". 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.

Faster display updates occur if text uses a solid background colour (ie no alpha blending).

Examples:
```
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)
```

Editable Text and Visible Cursor
A text can contain single byte hex of the form \\00 to \\FF
A text can contain hidden codes for use in password and editable fields.
\\01 defines the text as a PASSWORD so that only ***** are shown.
\\02 defines a hidden cursor and \\03 a hidden cursor with insert ON
\\04 defines an underline cursor and \\05 an underline cursor with insert ON
\\06 defines a block cursor and \\07 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 its current location.
The CALC() command can then be used to manipulate the text and cursor in EditBox.

Example Editable Text:
TEXT(EditBox,"Hello\\04World",8ptTextRed); this places an underline cursor at W

TEXT Styles
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 www.itrontft.com.
Unique Font Overlay
It is possible to overlay one font over another to enable single byte operation with ASCII from \\20 to \\7F and Cyrillic, Greek, Hebrew, Bengali, Tamil, Thai or Katakana from \\80 to \\FF. The LIB() command is used to load the extended font at \\0080 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 \\80 to \\FF.

```
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.
   back=black;              //ONLY USE where a page has a style updated=changed;
   opacity = n;             // n = 0..100 where 0=transparent..100=opaque (default=100)
   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
   justify=left;            //justification of multi-line text left, right, centre. default=left (From V00.49.14)
   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
```

Using PAGE STYLE Update=Changed
To gain the faster refreshing, a few rules apply.
1/ Only the screen area where the changed entity is located is redrawn.
2/ The entity is redrawn on top of any existing pixels being displayed in that area.
3/ Entities with transparent backgrounds will show all previous rendering at that location in the transparent area.
4/ Hiding an entity will not produce any visible difference until a full page refresh is performed.
To support the "update=changed;" method.
1/ Do not use images with transparent backgrounds
2/ Specify the "back" colour in the style for text.
3/ To hide an entity, a "masking" image will need to be placed over the entity.
4/ To refresh only the entities on a page that have been modified, use the double semi-colon ";;" refresh method after the last entity, e.g. TEXT (txt1, "Hello");;.
5/ To refresh the whole page, use the SHOW(page); method.

Operational

DRAW(Name,X,Y,Style)

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.
Faster display updates occur if draw uses a solid background colour (ie no alpha blending).

DRAW accepts VARs, signed/unsigned integers
(U8, U16, U32, S8, S16, S32), floats (FLT) and pointers (PTR)

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 > "Name"); to set a pointer.

Example Draw

```
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.

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.

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
```

Graphs
A number of graph styles now exist as draw types:

```
  type=p; type=pixel;     // Pixel Scatter - places a point at x,y
  type=t; type=trace;     // Trace/Line - joins the dots between current point and previous point.
  type=y; type=yBar;      // Bar Y - draws vertical line from 0 to y and clears from y+1 to ymax
  type=x; type=xBar;      // Bar X - draws horizontal line from 0 to x and clears from x+1 to xmax

 The origin on the graph can be changed
   xOrigin=val; // (default=0)
   yOrigin=val; // (default=0)

 The scaling of pixels can be set:
   xScale=val; // (default=100.0) [val can be float and is a percentage]
   yScale=val; // (default=100.0) [val can be float and is a percentage]
Note to draw graph with 0,0 at top and n,n at bottom, use yScale=-100;

The graph can be made to scroll (currently right-to-left only supported)
  xScroll=val; // where val=0 (default - no scroll); val=n (scroll left n pixels before each plot
```

Please refer to the ADC analogue input section for a graph application example.

Draw Styles

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.

```
STYLE(gstyle,DRAW)  {
   type=trace; //The shape to draw. type = B/Box; C/Circle;  L/Line;, T/Trace; P/Pixel; y/yBar; x/xBar;
   maxX=100;  // Not required except for dynamic rotation where the maximum width is declared
   maxY=100;  // Not required except for dynamic rotation where the maximum height is declared
   col=green;  //Specify the line border colour of the shape. Use hex, colour name + alpha
   back=black; //Specify the fill colour of the shape. Use hex, colour name + alpha
   opacity = n;  // n = 0..100 where 0=transparent..100=opaque (default=100)
   width=3;    //Specify the line border width of the shape default = 1
   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 etc.
   xOrigin=50;   //specify graph x origin wit respect to declared graph
   yOrigin=50;  //specify graph y origin wit respect to declared graph
   xScale=200; //scale the value automatically to fit the graph
   yScale=200; //scale the value automatically to fit the graph
   xScroll=1;    //define scroll direction and increment 1=left to right one pixel, 0=none, -1=right to left
   }
```

Operational Box/Circle/Line v39 Graph = v47

IMG(Name,Source,Style)

Draw or update an Image. 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 or NAND flash, 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.
LIB can be used with BMP and JPG although due to the lossy nature of jpeg, it is used for non transparency images like backgrounds

Example:
IMG(MyPic,TopBtn,MyImage); **//**previously stored as TopBtn using LIB command
IMG(MyPic,"sdhc/TopBtn.bmp",90,60,MyImage); **//**stored on SDHC card

Image Styles
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.
  maxX=160;         // Not required except for dynamic rotation where the maximum width is declared
  maxY=40;          // Not required except for dynamic rotation where the maximum height is declared
  rotate=0;         // Specify the rotation of the shape with respect to the screen. 0,90,180,270
  action =i;        // defines the way in which an image is presented on screen
  step=20;          //sets the number of pixels an image moves when the action is a moving. 1-minimum of TFT screen's x or y.
  opacity = n;      // n = 0..100 where 0=transparent..100=opaque (default=100)
  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
```

If maxX and maxY are the same size as the loaded file or unspecified, the library image is used rather than a copy created which saves memory space. 24 bit images are stored as 32 bit data. 16 bit images are stored as 16 bit and only expanded to 32 bit during page refresh so optimizing memory usage.

Actions
* The way in which an image is displayed can be changed for slideshows.
STYLE(imgSt,Image){ action=type; step=pixels; }
> action type options are:
- i or instant = Instant (default);
- u or up = Move Up;
- d or down = Move Down;
- l or left = Move Left;
- r or right = Move Right;
- ur or ru or upright = Move Diagonal Up-Right
- dr or rd or downright = Move Diagonal Down-Right
- ul or lu or upleft = Move Diagonal Up-Left
- dl or ld or downleft = Move Diagonal Down-Left
- a or all = Sequence through all (except instant);

Using PAGE STYLE Update=Changed
To gain the faster refreshing, a few rules apply.
1/ Only the screen area where the changed entity is located is redrawn.
2/ The entity is redrawn on top of any existing pixels being displayed in that area.
3/ Entities with transparent backgrounds will show all previous rendering at that location in the transparent area.
4/ Hiding an entity will not produce any visible difference until a full page refresh is performed.

To support the "update=changed;" method.
1/ Do not use images with transparent backgrounds
2/ Specify the "back" colour in the style for text.
3/ To hide an entity, a "masking" image will need to be placed over the entity.
4/ To refresh only the entities on a page that have been modified, use the double semi-colon ";;" refresh method after the last update object, e.g. IMG(imgt1,myimg);;.
5/ To refresh the whole page, use the SHOW(page); method.

Operational

KEY(Name,Function,X,Y,Style)

Create a Touch Area of size X,Y or define a Key on the external keyboard.

The touch area can have a One Touch function by using the built in style TOUCH or TOUCHR (repeat)
More sophisticated function is available on touch change with TOUCHC
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 action=DOWN; and the other with action=UP;.

When specifying an external key action, the values for X and Y indicate the contact points on the key board matrix where K0 is \\00 through to K23 which is \\17 .
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 INT(...);

The last touch co-ordinates are stored in predefined variables TOUCHX and TOUCHY

The touch screen can be calibrated using the command SETUP(system) { calibrate=y; }
The position of touch keys can be temporarily viewed as a grey area using
SETUP(system) { test=showTouchAreas; } and hidden again using test=hideTouchAreas.
See the SYSTEM command for global touch screen debounce, sampling and accuracy parameters.

KEY(name,func,width,height,style); now accepts ints and vars for width and height. v47

* Restriction: If processing a function called from a KEY() command then further key presses will be ignored. Each touch key press function must be processed to completion before another can be processed. Please refer to the project example 'keyboard' for the technique to process keys.

Examples KEY
KEY(TopKey,TopFnc,90,50,MyTouch); a touch area 90x50 pixels. Create your own style MyTouch
KEY(ExtKey,ExFunc,\\07,\\10,MyIOK); This external key operates when K7 and K16 connect. Create your own style MyIOK {type=keyio}
KEY(ExtKey,ExFunc,K07,K16,MyIOK); This external key operates when K7 and K16 connect. Create your own style MyIOK {type=keyio}
KEY(TKey,[HIDE(SPage);SHOW(TPage);],50,50,TOUCH); Inline commands instead of function

Plan: KEY(ExtKey,ExFunc,K07,K16,PushKey); This external key operates when K7 and K16 connect.

KEY Styles
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.

```
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 and C for change. See note below
     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
```

Action Types
Styles for touch keys action=u|d|c; (up|down|change) - where change detects key down and key up
Built in touch styles
- TOUCH with type=touch; debounce=50; repeat1=0; repeat2=0; action=D;
- TOUCHR with type=touch; debounce=50; repeat1=1000; repeat2=200; action=D;
- TOUCHC with type=touch; debounce=50; repeat1=1000; repeat2=200; action=C;

a) KEY(name,func,width,height,style);
- supports existing implementation plus must be used for external keys
- 'func' is called for key down, up and repeat, depending on key style action
b) KEY(name,downFunc,upFunc,width,height,style);
- 'downFunc' called when key down detected and for key repeat, depending on key style action
- 'upFunc' called when key up detected, depending on key style action
- either 'downFunc' and/or 'upFunc' can be omitted if no function call required
c) KEY(name,[downFunc],[upFunc],[repFunc],width,height,style);
- 'downFunc' called when key down detected, depending on key style action
- 'repFunc' called when key up detected, depending on key style action
- 'upFunc' called when key up detected, depending on key style action
- either 'downFunc' and/or 'upFunc' and/or 'repFunc' can be omitted if no function call required

* Note external keys still only support actions of up and down and command KEY(name,func,x,y,style);

* Examples
KEY(key1,[LOAD(rs2,"a");],90,84,TOUCH); - 'a' is output on key down only
KEY(key2,[LOAD(rs2,"b");],90,84,TOUCHR); - 'b' are output on key down and key repeat
KEY(key3,[LOAD(rs2,"c");],90,84,TOUCHC); - 'c' are output on key down, key repeat and key up

KEY(key4,[LOAD(rs2,"d");],[LOAD(rs2,"e");],90,84,TOUCHC);- 'd' are output on key down and key repeat, 'e' is output on key up
KEY(key5,[LOAD(rs2,"f");],[LOAD(rs2,"g");],[LOAD(rs2,"h");],90,84,TOUCHC);- 'f' is output on key down, 'h' on key repeat, 'g' on key up
KEY(key6,,[LOAD(rs2,"i");],[LOAD(rs2,"j");],90,84,TOUCHC);- 'j' are output on key repeat, 'i' on key up

KEY(key7,,,[LOAD(rs2,"k");],90,84,TOUCHC);- 'k' are output on key repeat only
KEY(key8,,[LOAD(rs2,"l");],,90,84,TOUCHC);- 'l' is output on key up only

Resistive Touch Panel
Set up parameters for resistive touch panels are implemented in SETUP(TOUCH)

```
SETUP(TOUCH)
{
type = res; // default is 'res' for resistive touch
enable = y; // enable touch keys globally with y or n
samples = 20; // define the number of touch samples per interrupt. Defaults:4.3" = 12; 5.7" = 12; 7" = 22;
debounce = 10; // define the time period between each sampling period. Defaults: 4.3" = 25; 5.7" = 30; 7" = 25;
accuracy = 20; // define the 0.25 pixel accuracy of the samples. Defaults: 4.3" = 50; 5.7" = 14; 7" = 12;
inactive=1000; // time in milliseconds (0=off)
}

* Touch Inactive  - Added interrupt capability for when touch screen has been inactive (ie not touched) for a user settable duration.
LOAD(touch.inactive,500);

INT(name,TOUCHI,function);
```

Projective Capacitive Touch Panel Control

The projective capacitive touch for the TFT requires a capacitive touch controller and a capacitive touch panel which are connected to CN3 on the module which can be purchased from the Accessories page.

NOTE: These can only be used with itron Smart TFT modules where CN3 can be set to 3v3
4.3" PCB480272A Issue 8A or newer
5.7" & 7.0" PCB800480A Issue 3 or newer

The predefined touch types TOUCH, TOUCHR and TOUCHC all work with with the capacitive touch.
A setup is required to specify the structure of the touch screen used.

Example Code

```
SETUP(TOUCH)
{
type = cap; // default is 'res' for resistive touch
width = 480; // default to width of display
height = 272; // default to height of display
xnum = 19; // number of X electrodes
ynum = 11; // number of Y electrodes
gain = 0; // gain of ADC
threshold = 25; //
debounce = 3; //
address = 75; // I2C address of controller
inactive=1000; // time in milliseconds (0=off)
}

STYLE( WhitePg, Page) { back=white; } //White Page Style
STYLE( textstyle, Text ) { font=Ascii16; col=black; maxLen=32; maxRows=1; curRel=CC; } //Black Text Style
STYLE( boxstyle, Draw) { type=Box; col=Black; back=White; width=1; curRel=CC; } //White Box black border

STYLE(MYTOUCH,key){type=touch; debounce=250; repeat=0; delay=0; action = d; curRel=CC; } //Touch style on action down

VAR(Num,0,S16); //Signed variable to be incremented

PAGE( page1, WhitePg )
{
POSN(240,136); TEXT(Number,Num,textstyle); //variable

POSN(160,136); DRAW(DownBox,80,80,boxstyle); //Down Box
TEXT(DownNum,"-",textstyle); //Down Text
KEY(DownKey,[CALC(Num,Num,1,"-");TEXT(Number,Num);;],80,80,MYTOUCH); //Down Function

POSN(320,136); DRAW(UpBox,80,80,boxstyle); //Up Box
TEXT(UpNum,"+",textstyle); //Up Text
KEY(UpKey,[CALC(Num,Num,1,"+");TEXT(Number,Num);;],80,80,MYTOUCH); //Up Function
}

SHOW(page1);
```


Operational

SHOW(Name)

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 be used to refresh a page if entities have changed.

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

Operational

HIDE(Name)

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.

Hide(Entity1, Entity2, Entity3...);; multiple hide entities then refresh current page

Operational

DEL(Name)

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.

Del(Entity1, Entity2, Entity3...); multiple delete entities

Delete is operational V17
The function RESET(DELETED) to free memory is planned

;; - page refresh

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");;

;; = SHOW(this_page);

Page Refreshing v47
A new mode has been added for pages which allows for either
1/ All entities to be redrawn on a page when a double semicolon (;;) refresh is encountered (default), or
2/ Only the entities that have been modified since last refresh to be redrawn when a double semicolon (;;) is encountered.
A SHOW(page); command will always redraw all the entities on a page.
STYLE(name,PAGE) { update=all; } // default: refreshes all entities on a ;;
STYLE(name,PAGE) { update=changed; } // refreshes only changed entities on a ;;
Operational

RUN(Name)

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,Func2,Func3...FuncN); or a pointer to a function RUN(func-ptr);

RUN(varcmd)
* Added support for running commands from a text variable.
This is useful when sending a SMART command over a serial link embedded in a user protocol.
It is then possible to dynamically create new entities and pages remotely from a host in a user protocol.
LOAD(cmd, "LOAD(RS2,1);LOAD(RS2,\\22Hello\\22);"); // cmd is a text variable.Use \\22 to insert double quotes
RUN(cmd); //Sends 1Hello via RS232 port
* Inline functions also supported: RUN([LOAD(RS2, "Hello");]);

Operational except User code TBD.

FUNC(Name) {...}

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.

EXIT(Name) - end functions
> EXIT(name); // exit nested loops/functions up to and including loop/function with name

* Examples:
> FUNC(fn1) { if(x=5?[EXIT(fn1);]); ..... } // exits function when x=5 without running rest of code
> FUNC(fn2) { LOOP(lp3,100){ LOAD(RS2,"*"); if(quit=1?[EXIT(fn2);;]);
// sends 100 *'s through RS2 unless quit is set to 1, then loop and the function are exited (A screen refresh occurs before the exit)

* Note, if the name provided in the EXIT(name); command does not exist in the current function/loop nesting, then all loops and functions are exited up to the top level. It is not possible to exit the page loop in this way.
* Restriction: If processing a function called from a KEY() command then further key presses will be ignored. Each touch key press function must be processed to completion before another can be processed. Please refer to the project example 'keyboard' for the technique to process keys.

Operational

[cmd(..); cmd(..);.....cmd(..);] - Inline Functions

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.

Without inline:

```
KEY(keyFlr15,floor15fnc,104,84,TOUCH);  //calls function floor15fnc

FUNC(floor15fnc)
  {
  LOAD(vReqd,15);      TEXT(txtCurFlr,"15");      RUN(fncGo);
  }
```

With inline:

```
KEY(keyFlr15, [ LOAD(vReqd,15); TEXT(txtCurFlr,"15"); RUN(fncGo); ],104,84,TOUCH);
```

Operational

LOOP(Name,Var1){.....}

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 exit a LOOP using the command EXIT(Name); Loops can be nested and used in PAGES or FUNCTIONS.

Examples:

```
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);}
```

Loop Example 1

```
  FUNC(fn1)
   {
   VAR(ii,0,U8);
   VAR(jj,0,U8);
   VAR(kk,0,U8);
   LOOP(lp0,10)
    {
    LOOP(lp1,10)
     {
     LOOP(lp2,10)
      {
      LOAD(RS2,ii,",",jj,",",kk,"\\0d");
      CALC(kk,kk,1,"+");
      }
     CALC(jj,jj,1,"+");
     }
    CALC(ii,ii,1,"+");
    }
  LOAD(RS2,"\\0d");
  }
```

RS2 Outputs: 0,0,0\\0d0,0,1\\0d\\0,0,2\\0d...9,9,9\\0d\\0d

Loop Example 2

```
KEY(k0,[LOOP(klp,10){LOAD(RS2,"*");}LOAD(rs2,"\\0d\\0a");],480,136,TOUCH);
```

RS2 outputs on key press: **********\\0d\\0a

* Restriction: If the LOOP() command is within a function called from a KEY() command then further key presses will be ignored. Each touch key press function must be processed to completion before another can be processed. Please refer to the project example 'keyboard' for the technique to process keys.

EXIT(Name) - end loops

```
> EXIT(name); // exit nested loops up to and including loop with name
```

* Examples:

```
> LOOP(lp1,FOREVER){ CALC(x,y,z,"+"); IF(x=5?[EXIT(lp1);]); } // exit loop when x=5
```

* Note, if the name provided in the EXIT(name); command does not exist in the current loop nesting, then all loops and functions are exited up to the top level. It is not possible to exit the page loop in this way.

Precautions when using LOOP() including "Array Error - Subscript Out Of Range" message

* At the start of each pass through a loop, a check is performed to see if a touch screen key is being pressed and, if it is, then the associated touch key function is called. Caution must be observed with the touch key function to not modify variables that are being used within the loop otherwise undesired results can occur which can be difficult to spot or result in an error message.

* Example 1 - Variables

```
 VAR( varX, 0, U8 );
 // We have a simple function...
 FUNC( fnTest1 )
 {
  LOAD( varX, 0 );
  LOOP( lpTest1, 10 )
  {
   // [Touch Keys are effectively tested here]
   LOAD( RS2, varX );
   CALC( varX, varX, 1, "+" );
  }
 }

 // In a page we have...
 KEY( kyTest1, [ LOAD( varX, 0 ); ], 100, 100, TOUCH );
 // Normally we would get 0123456789 sent out of the RS2 port each time fnTest1 is run
 // If however the key kyTest1 is pressed when the loop is being run then the output may be changed to 0123012345!
```

* Example 2 - Arrays

```
 VAR( varArr, 0, U8, 5 );
 VAR( varY, 0, U8 );
 // We have another simple function...
 FUNC( fnTest2 )
 {
  LOAD( varY, 0 );
  LOOP( lpTest2, 5 )
  {
   // [Touch Keys are effectively tested here]
   LOAD( RS2, varArr.varY );
   CALC( varY, varY, 1, "+" );
  }
```

```
}

// In a page we have...
KEY( kyTest2, [ LOAD( varY, 0 ); ], 100, 100, TOUCH );
KEY( kyTest3, [ LOAD( varY, 3 ); ], 100, 100, TOUCH );

// Normally we would get the contents of varArr.0 varArr.1 varArr.2 varArr.3 varArr.4 sent out of the RS2 port each time fnTest2 is run
// If however the key kyTest2 is pressed when the loop is being run then the output may be changed to varArr.0 varArr.1 varArr.0 varArr.1 varArr.2!
// Or, an error when kyTest3 is pressed giving varArr.0 varArr.1 varArr.3 varArr.4 ** Array Error - Subscript Out Of Range ** (ie varArr.5 doesn't exist!)
```

* Good coding practice
  > Make sure variables used in a loop are not modified from a touch key function (unless this is a desired action)
  > If a variable does need to be changed then set a 'flag' in the key function and test the flag in the page loop and make the change there instead.

Operational

INT(Name,Buffer,Function)

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.

Interrupts are available for counters and timers CNTMILLI...TIMER0. See relative section.

This is currently set to interrupt on each character received for the 'Buffer':
> RS2RXC = RS232 Receive Character
> RS4RXC = RS485 Receive Character
> AS1RXC = Async1 Receive Character
> AS2RXC = Async2 Receive Character
> DBGRXC = Debug Receive Character
> I2CRXC = I2C Receive Character

NOTE: The Buffer must be read to clear the interrupt otherwise the Function will keep getting called!

Example:

```
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
  }
```

Operational

LIB(Name,Source)

Store image, font, user font or user code file in the library.

Image and Fonts from an SD Card (Onboard Flash)
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.

Example LIB(myimage,"SDHC/backimg.bmp?back=\\000007"); **v0.21.**
LIB(myimage,"SDHC/backimg.bmp?back=\\000007&rotate=180&scale=75"); **v0.21.**
LIB(asc16x16fnt,"SDHC/asc16B.fnt?start=\\0020"); **v0.27**

JPG image handling
LIB(libImg1, "SDHC/image.jpg");
NOTE: Use of JPEG files instead of bitmaps can significantly decrease load time. However the lossy nature of jpeg may not provide accurate transparency capability and is therefore most suitable for backgrounds.
Fast start can be achieved by only loading the main menu images at start up then loading other images on demand and setting a flag to indicate they have been loaded.

Image and User Font loaded from a Serial Link TBD
Where the image or font is sent over a serial interface use the following command structure.

Examples LIB(myimage,"rs2/myimg.bmp?back=\\FFFFFF&rotate=180&scale=75");
LIB(myimage,"rs4/mypic.bmp?back=\\FFFFFF");
LIB(myfont,"spi/fnt?start=\\0000");

User Code TBD
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.

.BMP is operational v17.
User Compiled Code and User Font Array TBD

LOAD(Dest,Name,Name,....)

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.

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.

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
```

Text to Integer/Float
```
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. v0.36

Example Pointers
To set/change which entity the entity pointer is pointing to you use '>' instead of ','.
```
  LOAD( EntPtr1>"Var1" ); // Set EntPtr1 to point to Var1
  LOAD( EntPtr1>"Var1",num,"3" ); // Set EntPtr1 to point to Var123 (very power full not found in C)
```

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"
```

Change Setup Parameters
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
```

Write files to NAND v43
Files can also be transferred from SDHC or serial port to NAND flash

```
LOAD(NAND,"SDHC/TU480A.mnu"); //use in conjunction with FPROG

LOAD(NAND,"EXT/filename.ext?size=xxxx&timedate=yyyyyyyy");
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz  //data zzzzz sent via serial port
```
Reads xxxx bytes from the current serial port and copies the specified file to NAND setting the file date to yyyyyyyy which is a string with the same date/time format as used by the RTC

Plan in v43.2 to allow change of style parameters

VAR(Name,Value,Style)
+ pointer usage
+ non volatile parameter storage

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);

A range of 'built in' styles exist like U8,U16,U32,S8,S16,S32,FLT1,FLT2,FLT3,FLT4,TXT as shown in VAR styles below.
These can be appended with E for storage in 'non volatile' EEPROM as described below.

Example Numbers
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.

Example Pointers
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,...);
```

VAR Data Styles
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 up to 17 decimal places
                 // 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 17, 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
 }
```

Built In Styles (Add E for EEPROM types Example **FLT4E**)
The following pre defined 'built in' style names are available
U8/U8E - type = U8, **U16/U16E** - type = U16, **U32/U32E** - type = U32
S8/S8E - type = S8, **S16/S16E** - type = S16, **S32/S32E** - type = S32
PTR/PTRE - type = pointer, **TXT/TXTE** - type = TEXT, length=32
FLT1/FLT1E - type = float, decimal = 1, **FLT2/FLT2E** - type = float, decimal = 2
FLT3/FLT3E - type = float, decimal = 3, **FLT4/FLT4E** - type = float, decimal = 4

Operational

Arrays

Arrays are an efficient way to organise and access data. Providing a 4 dimensional capability allows users to store data for multiple pages, each containing multiple objects with associated text and image entities.

A typical application would be a multi-page soft key menu system allowing user editing of key labels and images. Multiple language support is easier to achieve rather than using pointers to a list of variables.

Arrays can be handled using a single name to simplify transfer to and from the host.
In this example a single command sends the array data to the RS2 port with STX, array length in 4 byte padded 0 Hex, the contents of the array "PArray" then ETX. Checksum can be added as required.
The length of the array will have been pre-defined during creation.

LOAD(RS2, "\\02", %H04%lenArray, %r%PArray, "\\03");

Arrays are defined using an extension to the VAR() command.
Each required dimension is passed as an additional parameter to the command.

VAR(name,init,type,size0); One-dimensional (or single dimension) array
VAR(name,init,type,size0,size1); Two-dimensional array
VAR(name,init,type,size0,size1,size2); Three-dimensional array
VAR(name,init,type,size0,size1,size2,size3); Four-dimensional array

Definitions
Array: A data structure consisting of a collection of elements (values), each identified by at least one index.
Index: A non-negative integer used to index a value in an array. Indices can also be called subscripts.
Element: A location in the array data structure which is used to store a value.
Dimension: The dimension of an array is the number of indices needed to select an element.

Indexing Arrays
Arrays use zero-based indexing, i.e. the first element of the array is indexed by 0.
For example, we define a 20 element array as:
VAR(A, 0, U8, 20);
Then the elements of the array are indexed A.0 through to A.19.
Array elements are accessed by separating the indices with a dot.
Single dimension with A.0 through to 4 dimensions with name.idx0.idx1.idx2.idx3

Future Releases
Arrays currently accept numeric values and individual characters.
The CALC() command currently only accepts single elements from an array, however it is intended to add the ability to process a range on elements of an array in the future. If this is a critical requirement for your project then please email us.
Structures will be achieved using U8 arrays with the user defining U8,U16,U32, FLT, TXT entities and the TFT will take care of handling each type according to its format.

Single Dimension Arrays
These are defined as
VAR(name, init, type, size0);

For example, to create an 8 element array, named A, storing U8 data and initial values of 0,
VAR(A, 0, U8, 8);

Accessing the elements of a single dimension array uses just a single subscript.
LOAD(ivar, A.4);
LOAD(A.x, ivar);

As an addition to this, all elements of the array can be loaded with a single value.
LOAD(A, 45);
LOAD(A, ivar);

The whole array can be passed to a serial port, text box, text variable or another array.
LOAD(rs2, A);
LOAD(tvar, A);
TEXT(txt, A);
LOAD(A1, A);
TEXT(txt, %h02%A);

When the array is passed, the elements are sent A.0 to A.7
The array can be loaded with the contents of a serial buffer.
LOAD(A, as1);
A series of elements of an array can be loaded.
LOAD(A, 1, 2, 3, \\04, 5, ivar, 7, 8);

The existing text variables are like a single dimension array of 32 characters when using style TXT.

Two-Dimensional Arrays
These are defined as
VAR(name, init, type, size0, size1);

This can be pictured as a table with size0 rows and size1 columns.
For example, to create a 2 (row) by 32 (column) element array, named B, storing S16 data and initial values of 0,
VAR(B, 0, S16, 2, 32);

This type of array could be used to store 32 points (x,y) for a graph, row 0 holds x values, row 1 holds y values.
Accessing the elements of a two-dimensional array uses two subscripts.
LOAD(ivar, B.1.12);
LOAD(B.x.pt, ivar);

As an addition to this, all elements of the array can be loaded with a single value.
LOAD(B, 45);

LOAD(B, ivar);

A single row can be loaded with a value by specifying only the first (row) subscript.
LOAD(B.1, ivar);

The whole array can be passed to a serial port, text box, text variable or another array.
LOAD(rs2, B);
LOAD(tvar, B);
TEXT(txt, B);
LOAD(B1, B);
TEXT(txt, %h02%B);
When the whole array is passed, the elements are sent a row at a time,
i.e. B.0.0, B.0.1, B.0.2, ..., B.0.31, B.1.0, B.1.1, B.1.2, ..., B.1.31

A single row can be passed by specifying only the first (row) subscript.
LOAD(rs2, B.0);
When the single row is passed, the elements are sent B.0.0 to B.0.31

The array (or row of) can be loaded with the contents of a serial buffer.
LOAD(B, as1);
LOAD(B.1, rs2);

A series of elements of an array can be loaded.
LOAD(B.0, 1, 2, 3, \\04, 5, ivar, 7, 8);

A graph can automatically be plotted when passed a two-dimensional array in the format B.2.n, where n is the number of points and row 0 contains the x-values, row 1 the y-values.
DRAW(graph, B.0, B.1);

Three-Dimensional Arrays
These are defined as
VAR(name, init, type, size0, size1, size2);

This could be used to store data for three graphs each containing 50 (x,y) points.

For example, to create a 3 by 2 by 50 element array, named C, storing U8 data and initial values of 0,
VAR(C, 0, U8, 3, 2, 50);

Accessing the elements of a three-dimensional array uses three subscripts.
LOAD(ivar, C.1.1.5);
LOAD(C.g.x.pt, ivar);
As an addition to this, all elements of the array can be loaded with a single value.
LOAD(C, 45);
LOAD(C, ivar);

A single dimension can be loaded with a value by specifying only the first two subscripts.
LOAD(C.1.0, ivar);
This loads C.1.0.0 to C.1.0.49 with ivar.

Two dimensions can be loaded with a value by specifying only the first subscript.
LOAD(C.1, ivar);
This loads C.1.0.0 to C.1.1.49 with ivar.

The whole array can be passed to a serial port, text box, text variable or another array.
LOAD(rs2, C);
LOAD(tvar, C);
TEXT(txt, C);
LOAD(B1, C);
TEXT(txt, %h02%C);

When the whole array is passed, the elements are sent as,
C.0.0.0, C.0.0.1, ..., C.0.0.49, C.0.1.0, ..., C.0.1.49, C.1.0.0, ..., C.1.0.49, C.1.1.0, ..., C.1.1.49, C.2.0.0, ..., C.2.0.49, C.2.1.0, ..., C.2.1.49

A single dimension can be passed by specifying only the first two subscripts.
LOAD(rs2, C.2.0);

The elements are sent C.2.0.0 to C.2.0.49

Two dimensions can be passed by specifying only the first subscript.
LOAD(rs2, C.2);
The elements are sent C.2.0.0 to C.2.0.49 then C.2.1.0 to C.2.1.49
The array (or dimension) can be loaded with the contents of a serial buffer.
LOAD(C, as1);
LOAD(C.1, rs2);
LOAD(C.1.1, i2c);

A series of elements of an array can be loaded.
LOAD(C.0.0, 1, 2, 3, \\04, 5, ivar, 7, 8);

Graphs can automatically be plotted when passed a two-dimensional array in the format C.g.2.n,
where n is the number of points and row 0 contains the x-values, row 1 the y-values.
DRAW(graph0, C.0.0, C.0.1);
DRAW(graph1, C.1.0, C.1.1);
DRAW(graph2, C.2.0, C.2.1);

Four-Dimensional Arrays
These are defined as
VAR(name, init, type, size0, size1, size2, size3);

For example, to create a 4 by 3 by 2 by 5 element array, named D, storing U8 data and initial values of 0,
VAR(D, 0, U8, 4, 3, 2, 5);

This could be used to represent touch panel game with 4-rows by 3-columns of buttons where each button
has a text string (maximum length 5 characters including terminating character) for the button when it is 'up' and when is 'down'.
The player has to 'guess' where the "YES!" is.

This can be pictured as
Buttons Up Buttons Down
"1" "2" "3" "no" "no" "no"
"4" "5" "6" "no" "no" "YES!"
"7" "8" "9" "no" "no" "no"
"10" "11" "12" "no" "no" "no"

Accessing the elements of a four-dimensional array uses four subscripts.
LOAD(ivar, D.2.1.1.4);
LOAD(D.r.c.u.s, ivar);

As an addition to this, all elements of the array can be loaded with a single value.
LOAD(D, 45);
LOAD(D, ivar);

A single dimension can be loaded with a value by specifying the first three subscripts.
LOAD(D.1.0.0, ivar);
This loads D.1.0.0.0 to D.1.0.0.4 with ivar.

Two dimensions can be loaded with a value by specifying the first two subscripts.
LOAD(D.1.2, ivar);
This loads D.1.2.0.0 to D.1.2.1.4 with ivar.

Three dimensions can be loaded with a value by specifying only the first subscript.
LOAD(D.1, ivar);
This loads D.1.0.0.0 to D.1.2.1.4 with ivar.

The whole array can be passed to a serial port, text box, text variable or another array.
LOAD(rs2, D);
LOAD(tvar, D);
TEXT(txt, D);
LOAD(B1, D);
TEXT(txt, %h02%D);

When the whole array is passed, the elements are sent as,
D.0.0.0.0, D.0.0.0.1, ..., D.3.2.1.4
A single dimension can be passed by specifying the first three subscripts.
LOAD(rs2, D.1.2.0);
TEXT(but1, %r%D.1.2.0);
The elements are sent D.1.2.0.0 to D.1.2.0.4

Two dimensions can be passed by specifying the first two subscripts.
LOAD(rs2, D.2.1);
The elements are sent D.2.1.0.0 to D.2.1.1.4

Three dimensions can be passed by specifying the first subscript.
LOAD(rs2, D.2);
The elements are sent D.2.0.0.0 to D.2.2.1.4

The array (or dimension) can be loaded with the contents of a serial buffer.
LOAD(D, as1);
LOAD(D.1, rs2);
LOAD(D.1.1, i2c);
LOAD(D.1.1.1, rs4);

A series of elements of an array can be loaded.
LOAD(D.0.0.0, 1, 2, 3, \\04, 0);

Array with Text Usage
This temporary solution enables text to be put in arrays using formatting %t%.
An intelligent text handling solution is being developed.

If we have the following variables:
VAR(A, \\ff, U8, 10);
VAR(u8Var, 63, U8);
VAR(txtVar1, "123XYZ", TXT);
VAR(txtVar2, "PQ", TXT);
VAR(txtVar3, "ABCDEFGHIJKLMN", TXT);

All examples assumed from initial declaration of array A

LOAD(A, 4); // A = { 4, 4, 4, 4, 4, 4, 4, 4, 4, 4 } <- fill array
LOAD(A.1, 3); // A = { \\ff, 3, \\ff, \\ff, \\ff, \\ff, \\ff, \\ff, \\ff, \\ff } <- fill single location
LOAD(A, u8Var); // A = { 63, 63, 63, 63, 63, 63, 63, 63, 63, 63 } <- fill array
LOAD(A.4, u8Var); // A = { \\ff, \\ff, \\ff, \\ff, 63, \\ff, \\ff, \\ff, \\ff, \\ff } <- fill single location

LOAD(A, txtVar1); // A = { 123, 123, 123, 123, 123, 123, 123, 123, 123, 123 } <- fill whole array with number
LOAD(A.2, txtVar1); // A = { \\ff, \\ff, 123, \\ff, \\ff, \\ff, \\ff, \\ff, \\ff, \\ff } <- single destination specified
LOAD(A.2, txtVar2); // A = { \\ff, \\ff, \\00, \\ff, \\ff, \\ff, \\ff, \\ff, \\ff, \\ff } <- unable to convert to a number so result is zero

LOAD(A.1, %t% txtVar1); // A = { \\ff, '1', \\ff, \\ff, \\ff, \\ff, \\ff, \\ff, \\ff, \\ff } <- single destination specified, no padding applied

```
LOAD( A, %t% txtVar1, %t% txtVar2 );  // A = {  '1',  '2',  '3',  'X',  'Y',  'Z',  'P',  'Q', \\00, \\00 } <- concatenation with zero padding

LOAD( A, %t% txtVar1 );     // A = {'1','2','3','X','Y','Z',\\00,\\00,\\00,\\00}
                                  <- text formatting is supplied, string is shorter than array dimension to pad with zeros

LOAD( A, %t% txtVar3 );    // A = {'A','B','C','D','E','F','G','H','I','J'} <- Longer than array, no terminator reqd as array dimension know
LOAD( A, %n% u8Var );  // A = {'6','3',\\00,\\00,\\00,\\00,\\00,\\00,\\00,\\00 } <- text formatter supplied, therefore store as text
LOAD( A, %H04% u8Var );  // A={'0','0','3','F',\\00,\\00, \\00, \\00, \\00, \\00 } <- text formatter supplied, therefore store as text


LOAD( A, "0x", %H04% u8Var ); // A = { \\00,  '0',  '0',  '3',  'F', \\00, \\00, \\00, \\00, \\00 }
                                  ** wrong - no text format provided to first string, therefore attemps conversion to number

LOAD( A, %t% "0x", %H04% u8Var );     // A = {  '0',  'x',  '0',  '0',  '3',  'F', \\00, \\00, \\00, \\00 } <- correct - text formatting applied


If A = { '1', '2', '3', 'X', 'Y', 'Z', \\00, 'P', 'Q', \\00 }
LOAD( RS2, %r% A ); \\ RS2 = "123XYZ\\00PQ\\00"    <- zeros are also sent
LOAD( RS2, %t% A ); \\ RS2 = "123XYZ" <- zeros are text terminators and are not sent, nor is text after a zero


Note:
The same destination values would be achieved
if TEXT( txt, %r% A );
or TEXT( txt, %t% A );
or LOAD( txtVar, %r% A );
or LOAD( txtVar, %t% A );
```

Array Minimum and Maximum Values

```
CALC( ... "MIN" ) and CALC( ... "MAX" )
* New functions added to obtain the minimum and maximum values stored in an array
  > CALC( val, array, "MIN" );  The minimum value in the 'array' is stored in 'val'
  > CALC( val, array, "MAX" );  The maximum value in the 'array' is stored in 'val'
```

Array Data Shift

```
CALC( ... "SHIFT" )
* New function added to shift the values in an array up or down its indices.
  > CALC( array, carry, shift, "SHIFT" );
  > CALC( array, shift, "SHIFT" );
- 'array' is shifted by 'shift' places, one shift at a time.
- If 'shift' is positive then array.1 -> array.2; array.0 -> array.1 etc
- If 'shift' is negative then array.1 -> array.0; array.2 -> array.1 etc
- The shift is 'circular', so the shifted out value is shifted in the other end.
- If 'carry' is specified then the shift passes through the carry, ie the carry is shifted in and a new
  carry is shifted out.

- Example
  VAR( arr, 0.0, FLT2, 5 );
  VAR car, 8.88, FLT2 );
  LOAD( arr.0, 0.00 );
  LOAD( arr.1, 1.11 ); LOAD( arr.2, 2.22 );
  LOAD( arr.3, 3.33 ); LOAD( arr.4, 4.44 );    // setup arr = { 0.00, 1.11, 2.22, 3.33, 4.44 }

  CALC( arr, -1, "SHIFT" );                 // Gives arr = { 1.11, 2.22, 3.33, 4.44, 0.00 }
  CALC( arr, 2, "SHIFT" );                  // Gives arr = { 4.44, 0.00, 1.11, 2.22, 3.33 }
  CALC( arr, car, -1, "SHIFT" );             // Gives arr = { 0.00, 1.11, 2.22, 3.33, 8.88 } and car = 4.44


  LOAD( car, 9.99 );
  CALC( arr, car, -1, "SHIFT" );       // Gives arr = { 1.11, 2.22, 3.33, 8.88, 9.99 } and car = 0.00
```

SWITCH / SELECT - CASE method

It is possible to emulate the SELECT CASE or SWITCH CASE function found in other languages.
This is used to test the contents of a variable and selectively process data according to its value.

The method used in the Itron TFT modules is different since it can directly jump to functions located anywhere in the program provided they use a common naming method.

It makes use of the ability to compile a function name in a variable and then use the RUN(variable); command
The example combines "case_" with "DC" to form a function name "case_DC".

The program can then contain functions to serve all the input options

This removes the need for multiple IF statements.

A typical 'c' example is as follows

```
switch ( input )
     {
        case "DC":
           DCfunc();
           break;
        case "DCT":
           DCTfunc();
           break;
        case "C":
           Cfunc();
           break;
        case "D":
           Dfunc();
           break;
      default :
           defunc();
     }
```

```
public DCfunc() {.....}
public DCTfunc() {.....}
public Cfunc() {.....}
public Dfunc() {.....}
public defunc() {.....}
```

The equivalent method is shown below

```
LOAD(chkstr , "," , input , ",");
CALC(tmp, ",C,D,DC,DCT," , chkstr, "FIND");
IF(tmp< 0 ? case_default : [LOAD(input,"case_",input); RUN(input);]);
```

This 3 line technique adds "," to front and end of the input value and loads into chkstr
If input="DC" then chkstr=",DC,"

A CALC command compares the chkstr with a list to identify if the input value exists
The existing commands are defined by ",C,D,DC,DCT,"

If tmp is -1 the input does not exist and the default function "case_default" is RUN.

If tmp is 0+n, the command exists and a prefix is added to input and RUN.
Where input = "DC", it exists so the function name "case_DC" is created and RUN

tmp and chkstr are predefined variables type U8 and TXT.

```
FUNC(case_DC) {.....}
FUNC(case_DCT) {.....}
FUNC(case_C) {.....}
FUNC(case_D) {.....}
FUNC(case_default) { .....}
```

Formatting Text and Serial Data Output

Numbers can now be formatted for storing into text vars, text areas, and transmission from serial ports.
Uses %fomat% in front of a variable.

The following examples use a variable VAR containing a value of 2031 and VARF containing 3.1415927

Decimal - s or no format supplied
> text(tx,VAR);; -> shows "2031"
> load(rs2,%%VAR); -> outputs "2031"
> var(txVar,%s%VAR); -> stores "2031"

Hex - h or H to store variable as hex where h=lowercase a-f, H = uppercase A-F.
> text(tx,%h%VAR);; -> shows "7ef"
> text(tx,%H%VAR);; -> shows "7EF"

Use h1 to h8 and H1 to H8 to store a variable with a field width where padding uses spaces
> text(tx,%H8%VAR);; -> shows " 7EF"
> text(tx,%h2%VAR);; -> shows "7ef"

Use h01 to h08 and H01 to H08 to store a variable with a field width where padding uses 0's
> text(tx,%H08%VAR);; -> shows "000007EF"
> text(tx,%h02%VAR);; -> shows "7ef"

Float - f to store variable as float
> text(tx,%f%VARF);; -> shows "3.141593"

f1 to f8 to store variable with number of decimal places
> text(tx,%f4%VARF);; -> shows "3.1416"
> text(tx,%f8%VARF);; -> shows "3.14159270"

Raw - r to store variable as raw number
> text(tx,%r%51);; -> shows "3"

C-Library printf format - * followed by standard C-library printf() formatting parameters
> text(tx,%*08X%VAR);; -> converts to "%08X" and shows "000007EF"
> text(tx,%*+d%VAR);; -> converts to "%+d" and shows "000007EF"
> text(tx,%*e%VARF);; -> converts to "%e" and shows "3.141593e+00"

IF(Var~Var?Function1:Function2)

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.
When comparing floating point numbers (max 17 decimal places) the lowest bit is masked prior to comparison.

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

Examples:

```
IF(K0=“L”?HELPFNC);     //single condition
IF(HIGHVAL < ACTVAL ? HIGHFUNC : LOWFUNC);
IF(STRVAR~>0? SHOWFUNC);   //if STRVAR length > 0 show data
IF(STARVAL >= -STARTMP?SHOWSTAR);
IF(STARVAL > 0? [ LOAD(vReqd,15); TEXT(txtCurFlr,"15"); RUN(fncGo); ] ); //uses in line code [..]
```

Operational v18

CALC(Result,VarA,VarB,Method)
CALC is used for numerics, maths, trigonometric, text and buffer manipulation plus file handling and checksums.

Numeric Handling
This provides a fast simple calculation placed in the Result variable according to the type
of Method using + , - , / , *, %(modulus) or logical functions | (OR) & (AND) ^ (EXOR) for non float.
The source parameters can be text, numeric, variables or pointers as appropriate.
More complex calculation methods for maths, trigs, text and buffers are described below.

Maths Functions
ABS - Absolute Value of varX - CALC(varD, varX, "ABS");
EXP - Exponential Function of varX - CALC(varD, varX, "EXP");
LOG - Natural Logarithm of varX - CALC(varD, varX, "LOG");
LOG10 - Base-Ten Logarithm of varX - CALC(varD, varX, "LOG10");
POW - varX Raised to the Power of vary - CALC(varD, varX, varY,"POW");
SQRT - Non-Negative Square Root of varX - CALC(varD, varX, "SQRT");
CBRT - Cube Root of varX - CALC(varD, varX, "CBRT");

Trigonometric Functions:
varD is result, varX is source, set system parameter angle=degres or radians;
COS - Cosine of varX - CALC(varD, varX, "COS");
SIN - Sine of varX - CALC(varD, varX, "SIN");
TAN - Tangent of varX - CALC(varD, varX, "TAN");

ACOS - Arc Cosine of varX - CALC(varD, varX, "ACOS");
ASIN - Arc Sine of varX - CALC(varD, varX, "ASIN");
ATAN - Arc Tangent of varX - CALC(varD, varX, "ATAN");
ATAN2 - Arc Tangent of varX/varY - CALC(varD, varX, varY, "ATAN2");

COSH - Hyperbolic Cosine of varX - CALC(varD, varX, "COSH");
SINH - Hyperbolic Sine of varX - CALC(varD, varX, "SINH");
TANH - Hyperbolic Tangent of varX - CALC(varD, varX, "TANH");
ACOSH - Hyperbolic Arc Cosine of varX - CALC(varD, varX, "ACOSH");
ASINH - Hyperbolic Arc Sine of varX - CALC(varD, varX, "ASINH");
ATANH - Hyperbolic Arc Tangent of varX - CALC(varD, varX, "ATANH");

Text and Cursor Handling
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

Cursor and Text Types
\\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

Text and Cursor Method Types - The first character in a string is position 0.
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 characters from the start of string as specified in VarB
RTRIM Remove characters 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 first case sensitive text VarB in VarA
LFIND Result gives the start position of last case sensitive text VarB in VarA
IFIND Result gives the start position of first case insensitive text VarB in VarA
ILFIND Result gives the start position of last 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

Buffer Handling
Buffer Method Types - for use with raw data. The first byte is position 0.
BCOPY Buffer Copy - Copy n bytes from start, end or position
BCUT Buffer Cut - Cut n bytes from start, end or position
BINS Buffer Insert - Insert bytes at position
BREP Buffer Replace - Replace bytes at position
BFIND Buffer Find - Locate first data from position
BLFIND Buffer Find - Locate last data from position
BLEN Get Buffer Length - Calculate number of bytes
BTRIM Buffer Trim Start and End - Remove bytes beginning and end
BLTRIM Buffer Trim Start - Remove bytes from start
BRTRIM Buffer Trim End - Remove bytes from end

BREM Buffer Remove - Find and remove bytes

User Protocol Split
MSPLIT Perform a multiple split of a buffer to a series of variables.

File Handling
FEXISTS Checks for existence of file in NAND or on SDHC cardCALC(dstVar, src, "FEXISTS");
FREAD A text file can be read into a text buffer CALC(dstTxtVar, src, "FREAD");
DIR NAND directory listing loaded into text variable with comma separation options

A detailed description follows for each method.
'dst' and 'src' can be the same text variable.
'src' is unmodified unless same text variable as 'dst'.
Supported data types:
The parameters can be text, numeric, variables or pointers as appropriate

"POS" - Move Cursor to Absolute Position
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 the length of 'src' then the cursor is placed after the last character.

"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.

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

"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'.

"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.

"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.

"RTRIM" - Trim Characters from End of Text String
CALC(dst, src, list, "LTRIM");
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.

"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'.

"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'.

"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 in 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".

"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 in 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".

"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.
If 'type' is a string then the first character is taken as the cursor type.
'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.

"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.

"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.

"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.

"LFIND" - Find Location of Text1 in Text2
CALC(loc, src1, src2, "FIND");
The last 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.

"IFIND" - Find Location of Case Insensitive Text1 in Text2
CALC(loc, src1, src2, "FIND");
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.

"ILFIND" - Find Location of Case Insensitive Text1 in Text2
CALC(loc, src1, src2, "FIND");
The last 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.

"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'.

"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'.

"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. See BCUT for similar function where length is used instead of delimiter.

"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.

"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.

Buffer Copy - Copy n bytes
CALC(dst,src,len,"BCOPY"); Copy length from start of src or if length negative, from end
CALC(dst,sc,pos,len,"BCOPY"); Copy length from posn of src or if length negative, before posn

Buffer Cut - Cut n bytes
CALC(dst,src,len,"BCUT"); Cut length from start of src or if length negative, from end and put in dst. src is modified
CALC(dst,src,pos,len,"BCUT"); Cut length from posn of src or if length negative, before posn and put in dst. src is modified

Buffer Insert - Insert bytes at position
CALC(dst,src,insvar,pos,"BINS"); Insert insvar at posn of src

Buffer Replace - Replace bytes at position
CALC(dst,src,insvar,pos,"BREP"); Replace insvar over src content from pos of src

Buffer Find - Locate data from position
CALC(dst,src,fvar,"BFIND"); Find first location of fvar content from pos of src

Buffer Find Last - Locate last data from position
CALC(dst,src,fvar,"BLFIND"); Find last location of fvar content from pos of src

Get Buffer Length - Calculate number of bytes
CALC(dst,src,num,"BLEN"); Dst is length of srcr + num

Buffer Trim Start and End - Remove bytes beginning and end
CALC(dst,src,trimvar,"BTRIM"); Remove bytes in trimvar from front and end of src

Buffer Trim Start - Remove bytes from start

CALC(dst,src,trimvar,"BLTRIM"); Remove bytes in trimvar from front of src

Buffer Trim End - Remove bytes from end
CALC(dst,src,trimvar,"BRTRIM"); Remove bytes in trimvar from end of src

Buffer Remove - Find and remove bytes
CALC(dst,src,remvar,"BREM"); Remove every occurance of remvar from src

NAND directory listing
Text variable can be populated with filenames in NAND:
CALC(dstTxtVar, src, filter, sep, "DIR");
CALC(dstTxtVar, src, filter, "DIR");
CALC(dstTxtVar, src, "DIR");

dstTxtVar is a text variable and will contain number of files in list followed by specified separator between each filename.
src is "nand" (in quotes)
sep is separator character for filenames in string (must be in quotes), if not included then "," assumed

filter is types of files to list, supported filters:
"*.*", "*",
"*.bmp", "*.jpg", "*.png" (or "img" to get all image files)
"*.wav", "*.mp3", "*.wma" (or "snd" to get all sound files)
"*.fnt",
"*.txt",
"*.mnu",
If not specified then "*" assumed.
Multiple filters can be included, separated by commas: "*.bmp,*.jpg,*.fnt"

Examples: (nand contains 1.bmp, 2.bmp, x.mnu)

CALC(txtVar, "nand", "*.bmp", ",", "DIR"); // list all bitmap image files
> txtVar = "2,1.bmp,2.bmp";
CALC(txtVar, "nand", "*.fnt", "DIR"); // list all font files
> txtVar = "0";
CALC(txtVar, "nand", "*.mnu", "DIR"); // list all menu files
> txtVar = "1,x.mnu";
CALC(txtVar, "nand", "DIR"); // list all files
CALC(txtVar, "nand", "*", "DIR"); // same
CALC(txtVar, "nand", "*", ",", "DIR"); // same
> txtVar = "3,x.mnu,1.bmp,2.bmp";

Checksums
* CALC(dst, src, "type", "MCHK"); // Copy src buf to dst buf, make checksum of "type" and append to dst buf
* CALC(res, src, "type", "TCHK"); // Test checksum of "type" in src buf and set res to 1 if checksums same, else 0.

where "type" is:
"SUM8ZA" - Sum all data in src as type U8, checksum is two's complement of the sum, stored as two ASCII hexadecimal characters
(when sum added to checksum is zero, res is 1)
"SUM8ZD" - Sum all data in src as type U8, checksum is two's complement of sum, stored as single U8
(when sum added to checksum is zero, res is 1)
"SUM8A" - Sum all data in src as type U8, checksum is sum, stored as two ASCII hexadecimal characters
(when sum is same as checksum, res is 1)
"SUM8D" - Sum all data in src as type U8, checksum is sum, stored as single U8
(when sum is same as checksum, res is 1)
"XOR8A" - Exclusive-OR (XOR) of all data in src as type U8, checksum is XOR, stored as two ASCII hexadecimal characters
(when XOR of src with checksum is zero, res is 1)
"XOR8D" - Exclusive-OR (XOR) of all data in src as type U8, checksum is XOR, stored as single U8
(when XOR of src with checksum is zero, res is 1)

* Example:
```
LOAD(txData,"1234");
FUNC(SendData)
{
CALC(txData,txData,"SUM8ZA","MCHK"); // Add checksum (txData="123436")
LOAD(RS2,txData); // Send data
}

FUNC(ReceiveData)
{
LOAD(rxData,RS2); // Read data
CALC(res,rxData,"SUM8ZA","TCHK"); // Check for checksum error
IF(res=1?ProcessData:DataError);
}
```

Text, Draw and Image Entity Information
* More Calc commands added to obtain entity information
> CALC(var,ename,"ESIZE"); -> returns allocated display size in bytes
> CALC(var,ename,"EDEL"); -> returns 1 if entity has been deleted, else 0
> CALC(var,ename,"EVIS"); -> returns 1 if entity is visible, else 0
> CALC(var,ename,"EALIGN"); -> returns value representing alignment:
0 = Top Left, 1 = Top Centre, 2 = Top Right,
3 = Centre Left, 4 = Centre Centre, 5 = Centre Right,
6 = Bottom Left, 7 = Bottom Centre, 8 = Bottom Right

CRC-16 Support - CALC(dst16, srcBuf, "type", "CRC16");
* Support for additional CRC-16 algorithms.
> Note CALC(dst32, srcBuf, "", "CRC16"); will use the MODBUS ("modbus") algorithm

CALC() "type"	Poly-nominal	Initial Value	Reflect In	Reflect Out	XOR Out Value	Names and aliases
"arc"	0x8005	0x0000	Yes	Yes	0x0000	"ARC", "CRC-16", "CRC-IBM", "CRC-16/ARC", "CRC-16/LHA"
"kermit"	0x1021	0x0000	Yes	Yes	0x0000	"KERMIT" "CRC-16/CCITT", "CRC-16/CCITT-TRUE", "CRC-CCITT"
"modbus"	0x8005	0xFFFF,	Yes	Yes	0x0000	"MODBUS"
"x-25"	0x1021	0xFFFF	Yes	Yes	0xFFFF	"X-25", "CRC-16/IBM-SDLC", "CRC-16/ISO-HDLC", "CRC-B"
"xmodem"	0x1021	0x0000	No	No	0x0000	"XMODEM", "ZMODEM", "CRC-16/ACORN"
"ccitt-f"	0x1021	0xFFFF	No	No	0x0000	"CRC-16/CCITT-FALSE"
"usb"	0x8005	0xFFFF	Yes	Yes	0xFFFF,	"CRC-16/USB"
"spi"	0x1021	0x1D0F	No	No	0x0000	"CRC-16/SPI-FUJITSU", "CRC-16/AUG-CCITT"
"buypass"	0x8005	0x0000	No	No	0x0000	"CRC-16/BUYPASS", "CRC-16/VERIFONE"
"dds-110"	0x8005	0x800D	No	No	0x0000	"CRC-16/DDS-110"
"dect-r"	0x0589	0x0000	No	No	0x0001	"CRC-16/DECT-R"
"dect-x"	0x0589	0x0000	No	No	0x0000	"CRC-16/DECT-X"
"dnp"	0x3D65	0x0000	Yes	Yes	0xFFFF	"CRC-16/DNP"
"en13757"	0x3D65	0x0000	No	No	0xFFFF	"CRC-16/EN-13757"
"genibus"	0x1021	0xFFFF	No	No	0xFFFF	"CRC-16/GENIBUS", "CRC-16/EPC", "CRC-16/I-CODE", "CRC-16/DARC"
"maxim"	0x8005	0x0000	Yes	Yes	0xFFFF	"CRC-16/MAXIM"
"mcrf4xx"	0x1021	0xFFFF	Yes	Yes	0x0000	"CRC-16/MCRF4XX"
"riello",	0x1021	0xB2AA	Yes	Yes	0x0000	"CRC-16/RIELLO"
"t10-dif",	0x8BB7	0x0000	No	No	0x0000	"CRC-16/T10-DIF"
"teledsk"	0xA097	0x0000	No	No	0x0000	"CRC-16/TELEDISK"
"tms371x"	0x1021	0x89EC	Yes	Yes	0x0000	"CRC-16/TMS37157"
"a"	0x1021	0xC6C6	Yes	Yes	0x0000	"CRC-A"

* Results have been confirmed using the "123456789" test with the values at http://regregex.bbcmicro.net/crc-catalogue.htm

CRC-32 Support - CALC(dst32, srcBuf, "type", "CRC32");

* Support for CRC-32 algorithms
> Note CALC(dst32, srcBuf, "", "CRC32"); will use the CRC-32 ("adcpp") algorithm

CALC() "type"	Poly-nominal	Initial Value	Reflect In	Reflect Out	XOR Out Value	Names and aliases
"adcpp"	0x04C11DB7	0xFFFFFFFF	Yes	Yes	0xFFFFFFFF	"CRC-32", "CRC-32/ADCCP", "PKZIP"
"bzip2"	0x04C11DB7	0xFFFFFFFF	No	No	0xFFFFFFFF	"CRC-32/BZIP2", "CRC-32/AAL5", "CRC-32/DECT-B", "B-CRC-32"
"c"	0x1EDC6F41	0xFFFFFFFF	Yes	Yes	0xFFFFFFFF	"CRC-32C", "CRC-32/ISCSI", "CRC-32/CASTAGNOLI"
"d"	0xA833982B	0xFFFFFFFF	Yes	Yes	0xFFFFFFFF	"CRC-32D"
"mpeg-2"	0x04C11DB7	0xFFFFFFFF	No	No	0x00000000	"CRC-32/MPEG-2"
"posix"	0x04C11DB7	0x00000000	No	No	0xFFFFFFFF	"CRC-32/POSIX", "CKSUM"
"q"	0x814141AB	0x00000000	No	No	0x00000000	"CRC-32Q"
"jamcrc"	0x04C11DB7	0xFFFFFFFF	Yes	Yes	0x00000000	"JAMCRC"
"xfer"	0x000000AF	0x00000000	No	No	0x00000000	"XFER"

* Results have been confirmed using the "123456789" test with the values at http://regregex.bbcmicro.net/crc-catalogue.htm

User Protocol Split

CALC(... "MSPLIT") Multi-Split

*Perform a multiple split of a buffer to a series of variables.
>>> CALC(dstPtr, srcBuf, char, "MSPLIT");

- The buffer 'srcBuf' is split at each 'char' and each result is stored in an incrementing series of variables prefixed with the name in 'dstPtr'.
- If 'dstPtr' contains "svar", the first variable will be 'svar0', then 'svar1', 'svar2', ..., 'svar9', 'svar10' etc.
- If a particular svarN is not defined then the result is not stored for that split.
- The data is stored in the format specified in each 'svarN' allowing the buffer to be split into text, unsigned/signed integers and floats.

- Example

```
VAR( dst > "svar", PTR );
VAR( svar0, 0, U8 );
VAR( svar1, 0, S32 );
VAR( svar2, 0.0, FLT4 );
VAR( svar3, "", TXT );
VAR( buf, "123,-67,3.14,Hi", TXT );
CALC( dst, buf, ",", "MSPLIT" );   // Gives: svar0 = 123, svar1 = -67, svar2 = 3.14, svar3 = "Hi"
```

>>> CALC(dstPtrArray, srcBuf, char, "MSPLIT");

- The buffer 'srcBuf' is split at each 'char' and each result is stored in the variables pointed to by successive subscripts of the Pointer Array 'dstPtrArray'.

- The first variable will be the variable name stored in 'dstPtrArray.0', then 'dstPtrArray.1', 'dstPtrArray.2',
- If a particular 'dstPtrArray.N' is not a variable or not defined then the result is not stored for that split.
- The data is stored in the format specified in each 'dstPtrArray.N' variable allowing the buffer to be split into text, unsigned/signed integers and floats.

- Example

```
VAR( arr > "", PTR, 4 );
VAR( alpha, 0, U8 );        LOAD( arr.0 > "alpha" );
VAR( bravo, 0, S32 );       LOAD( arr.1 > "bravo" );
```

```
VAR( charlie, 0.0, FLT4 );  LOAD( arr.2 > "charlie" );
VAR( delta, "", TXT );      LOAD( arr.3 > "delta" );
VAR( buf, "123,-67,3.14,Hi", TXT );
CALC( dst, buf, ",", "MSPLIT" );
// Gives: alpha = 123, bravo = -67, charlie = 3.14, delta = "Hi"
```

- Using this method with arrays of pointers, testing found the time to split 64 parameters using "MSPLIT" was 8ms compared to 64 individual "SPLIT"s which took 30ms.

Operational

Reserved Words - 17

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
DBGRXC	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 libary
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
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)
RTADAYS	numeric variable containing days (1-31)
RTAHOURS	numeric variable containing hours (0-23)
RTAMINS	numeric variable containing minutes (0-59)
RTAMONTHS	numeric variable containing month (1-12)
RTASECS	numeric variable containing seconds (0-59)
RTAYEARS	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
rxi	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
startup	show firmware version, progress bar or none at stat up
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)
TIMER0-TIMER9	timer variables
TOUCH	A preset style for TOUCH Key
TOUCHX	contains the last touch Y co-ordinate
TOUCHY	contains the last touch X co-ordinate
touchaccuracy	acceptance accuracy of samples
touchdebounce	time between sampling
touchsamples	number of samples per touch point
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
txi	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 List - 18

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

LOAD(ADC1.calib1,0.75); changes the calibration value for the analogue input ADC1.

Inherited Styles
* Style inheritance using previously defined style
> style(styleA,text){...}
> style(styleB,styleA){...} <- firstly copy style from styleA then apply new style parameters

Command	Description

VAR(Name,Value,Style)	**VAR Data Styles**
	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
	}
	
	Built In Styles (Add E for EEPROM types Example **FLT4E**)
	The following pre defined 'built in' style names are available
	U8/U8E - type = U8, **U16/U16E** - type = U16, **U32/U32E** - type = U32
	S8/S8E - type = S8, **S16/S16E** - type = S16, **S32/S32E** - type = S32
	PTR/PTRE - type = pointer, **TXT/TXTE** - type = TEXT, length=32
	FLT1/FLT1E - type = float, decimal = 1, **FLT2/FLT2E** - type = float, decimal = 2
	FLT3/FLT3E - type = float, decimal = 3, **FLT4/FLT4E** - type = float, decimal = 4
	
	Operational

PAGE(Name,Style) {.....}	**Page Styles**
	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.
	}
	
	Page with screen size or smaller.

TEXT(Name,Text,Style)	**TEXT Styles**
	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 www.itrontft.com.
	Unique Font Overlay
	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.
	
	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
	
	Operational

DRAW(Name,X,Y,Style)	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.
	Draw Styles
	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.
	
	STYLE(stCircleRed,DRAW)
	{
	type=B; //Specify the type of shape to draw. **type** = 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
	
	Operational

IMG(Name,Source,Style)	**Image Styles**
	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.
	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
	
	Operational

KEY(Name,Function,X,Y,Style)	**KEY Styles**
	Specify the source of key data. Touch keys are dependent on certain SYSTEM parameters
	If you require a dual action, specify 2 keys at the same location, one with action D and one with U.
	
	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 for the key.
	curRel=CC; //specify touch key placement relative to cursor. CC Centre Centre,
	} //BC Bottom Centre, LC Left Centre, RC Right Centre, TL Top Left,
	// BL Bottom Left, TR Top Right, BR Bottom Right, TC Top Centre.
	
	Operational.

Setup List - 19

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

System	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
	}

RS232	**Quick Setup**
	setup(RS2)
	{
	set="96NC" //quick set up combination "48, 96, 192, 384, 768, 1150 with parity N, O, E and Command //option".
	}
	
	Setup
	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
	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
	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
	}

RS485	**Quick Setup**
	setup(RS4)
	{
	set="96NC" //quick set up combination "48,96,192,384,768,1150 with parity N,O,E and Command option".
	}
	
	Setup
	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
	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
	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
	}

AS1, AS2, DBG	**Quick Setup**
	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
	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
	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
	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
	}

CANBUS 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, 15, 2  - note 15 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
}
```

SPI

Quick Setup

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

Setup

```
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
}
```

TWI / I2C

Quick Setup

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

Setup

```
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
}
```

KEY I/O

```
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"
}
```

PWM controller

```
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
}
```

ADC - A to D converters

```
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
}
```

Character Fonts

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

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
```

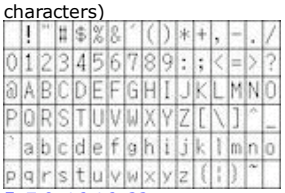
STANDARD ASCII - 20H to 7FH
Standard ASCII text in the range 20H to 7FH can by directly typed from the keyboard.
System fonts named ASCII8, ASCII16, ASCII32 are pre-installed.
Example TEXT(txt1, "Hello World", stTXT); //single byte access to 20H to 7FH ASCII characters

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. AB\\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. AB\\w00B0CDEF \\w00AB", stTXT );                  // UNICODE double byte access to 0080H to FFFFH
TEXT( txt3, "3. AB\\mC2B0CDEF \\mC2AB", stTXT );                  // UTF8 multi byte access to 80H to FFFFH
TEXT( txt5, "5. AB\\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<0038002e00200041004200B000430044004500460020 00AB>", 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

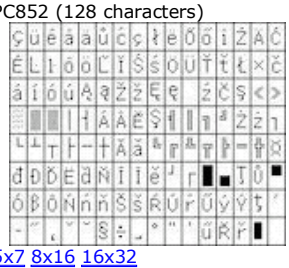
PC437 (128 characters)

5x7 8x16 16x32

PC850 (128 characters)

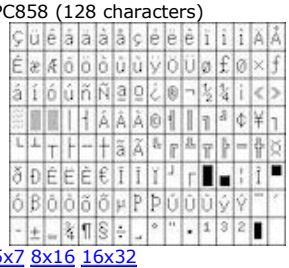
5x7 8x16 16x32

PC852 (128 characters)



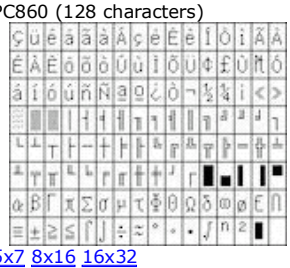
5x7 8x16 16x32

PC858 (128 characters)



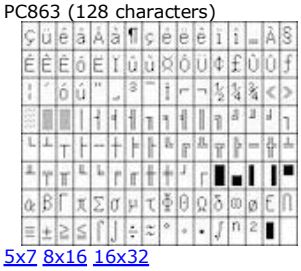
5x7 8x16 16x32

PC860 (128 characters)



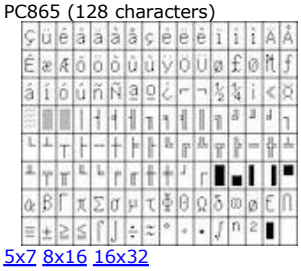
5x7 8x16 16x32

PC863 (128 characters)



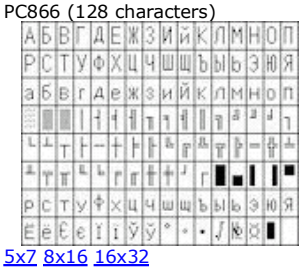
5x7 8x16 16x32

PC865 (128 characters)



5x7 8x16 16x32

PC866 (128 characters)

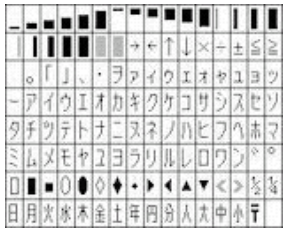


5x7 8x16 16x32

WPC1252 (128 characters)

5x7 8x16 16x32

Katakana (128 characters)



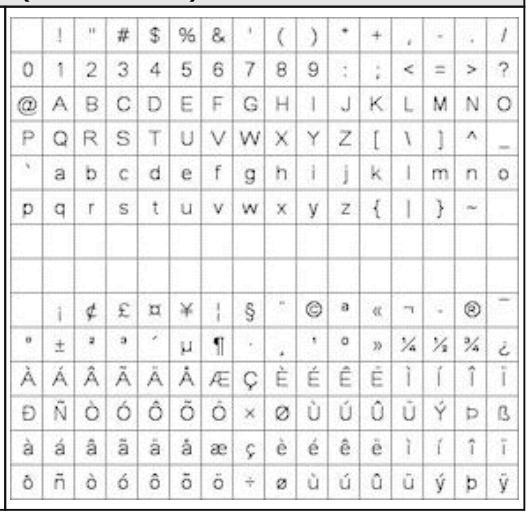
5x7 8x16 16x32

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



Cyrillic (226 characters)

Ъ
16px (3.2mm)
Ъ
24px (4.8mm)
Ъ
32px (6.4mm)

Unicode Range
0401 - 04F9

Greek (105 characters)

Ω

16px (3.2mm)

Ω

24px (4.8mm)

Ω

32px (6.4mm)

Unicode Range
0374 - 03F3

Arabic (194 characters)

پش

16px (3.2mm)

پش

24px (4.8mm)

پش

32px (6.4mm)

Unicode Range
060C - 06F9

Hebrew (82 characters)

ק

16px (3.2mm)

ק

24px (4.8mm)

ק

32px (6.4mm)

Unicode Range
0591 - 05F4

Bengali (89 characters)

আ

16px (3.2mm)

আ

24px (4.8mm)

আ

32px (6.4mm)

Unicode Range
0981 - 09FA

Tamil (61 characters)

ஹ

16px (3.2mm)

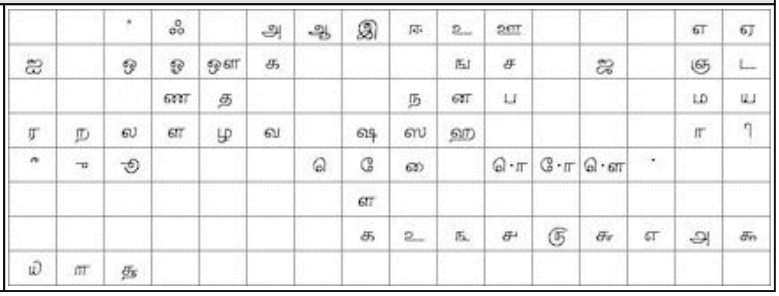
ஹ

24px (4.8mm)

ஹ

32px (6.4mm)

Unicode Range
0B82 - 0BF2



Thai (87 characters)

ส

16px (3.2mm)

ส

24px (4.8mm)

ส

32px (6.4mm)

Unicode Range
0E01 - 0E5B

Chinese/Japanese/Korean (21151 characters)

挑

16x16 (3.2mm)

挑

24x24 (4.8mm)

挑

32x32 (6.4mm)

Unicode Range
3300 - 9FA5

Hangul (11172 characters)

냿

16x16 (3.2mm)

냿

24x24 (4.8mm)

냿

32x32 (6.4mm)

Unicode Range
AC00 - D7A3

Katakana (94 characters)

ポ

16x16 (3.2mm)

ポ

24x24 (4.8mm)

ポ

32x32 (6.4mm)

Unicode Range
30A1 - 30FE

	ァ	ア	ィ	イ	ゥ	ウ	ェ	エ	ォ	オ	カ	ガ	キ	ギ	ク
グ	ケ	ゲ	コ	ゴ	サ	ザ	シ	ジ	ス	ズ	セ	ゼ	ソ	ゾ	タ
ダ	チ	ヂ	ッ	ツ	ヅ	テ	デ	ト	ド	ナ	ニ	ヌ	ネ	ノ	ハ
バ	パ	ヒ	ビ	ピ	フ	ブ	プ	ヘ	ベ	ペ	ホ	ボ	ポ	マ	ミ
ム	メ	モ	ャ	ヤ	ュ	ユ	ョ	ヨ	ラ	リ	ル	レ	ロ	ヮ	ワ
ヰ	ヱ	ヲ	ン	ヴ	ヵ	ヶ	ヷ	ヸ	ヹ	ヺ	・	ー	ヽ	ヾ	

Colour Chart - 21

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	#7CFC00 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	#FFEBCD blanchedalmond	#FFE4B5 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	#FFFFF0 ivory	#FFFAF0 floralwhite	#FAF0E6 linen	#FDF5E6 oldlace	#FAEBD7 antiquewhite
#FFE4C4 bisque	#FFDAB9 peachpuff	#FFEFD5 papayawhip	#F5F5DC beige	#FFF5EE seashell	#FFF0F5 lavenderblush	#FFE4E1 mistyrose	#FFFAFA 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

itron SMART TFTs - Program Basics

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 module is supplied with demonstration screens.

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 itron SMART 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 itron SMART 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
```

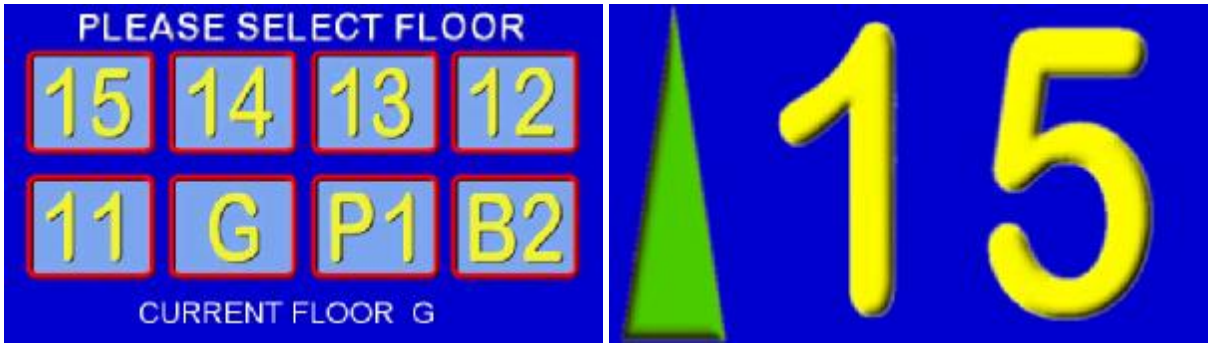
Hello World via RS232 IN with touch key to send RS232 OUT

```
// 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 )
{
POSN( 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 \\0D 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 )
{
POSN( 199, 136 ); IMG( LeftImg, Image1,172,240,StImg); // Position and draw 1st image on display
POSN( 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);
POSN( 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 TU800A.MNU for Elevator System using TU800X480C
// Updated 06-Nov-2010
// Floors are 15(15)..01(1), G(0), P1(-1), B2(-2)

// Load images into the library
LIB(libImgNum0,"SDHC/Lift08.bmp?back=\\0000CD"); // Load Number 0
LIB(libImgNum1,"SDHC/Lift18.bmp?back=\\0000CD"); // Load Number 1
LIB(libImgNum2,"SDHC/Lift28.bmp?back=\\0000CD"); // Load Number 2
LIB(libImgNum3,"SDHC/Lift38.bmp?back=\\0000CD"); // Load Number 3
LIB(libImgNum4,"SDHC/Lift48.bmp?back=\\0000CD"); // Load Number 4
LIB(libImgNum5,"SDHC/Lift58.bmp?back=\\0000CD"); // Load Number 5
LIB(libImgNum6,"SDHC/Lift68.bmp?back=\\0000CD"); // Load Number 6
LIB(libImgNum7,"SDHC/Lift78.bmp?back=\\0000CD"); // Load Number 7
LIB(libImgNum8,"SDHC/Lift88.bmp?back=\\0000CD"); // Load Number 8
LIB(libImgNum9,"SDHC/Lift98.bmp?back=\\0000CD"); // Load Number 9
LIB(libImgBChar,"SDHC/LiftB8.bmp?back=\\0000CD"); // Load Character B
LIB(libImgGChar,"SDHC/LiftG8.bmp?back=\\0000CD"); // Load Character G
LIB(libImgPChar,"SDHC/LiftP8.bmp?back=\\0000CD"); // Load Character P
LIB(libImgDTri,"SDHC/LiftDn8.bmp?back=\\0000CD"); // Load red triangle
LIB(libImgUTri,"SDHC/LiftUp8.bmp?back=\\0000CD"); // Load green triangle
LIB(libImgPMTD,"SDHC/LiftCls8.bmp"); // Load the warning message
LIB(libImgSelFlr,"SDHC/LiftSel8.bmp"); //Load the Select Floor Page
LIB(libImgDoors,"SDHC/LiftOpn8.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(104,158); KEY(keyFlr15,[LOAD(vReqd,15);TEXT(txtCurFlr,"15");RUN(fncGo);],135,100,TOUCH);
POSN(+194,+0); KEY(keyFlr14,[LOAD(vReqd,14);TEXT(txtCurFlr,"14");RUN(fncGo);],135,100,TOUCH);
POSN(+193,+0); KEY(keyFlr13,[LOAD(vReqd,13);TEXT(txtCurFlr,"13");RUN(fncGo);],135,100,TOUCH);
POSN(+192,+0); KEY(keyFlr12,[LOAD(vReqd,12);TEXT(txtCurFlr,"12");RUN(fncGo);],135,100,TOUCH);
POSN( 104,298); KEY(keyFlr11,[LOAD(vReqd,11);TEXT(txtCurFlr,"11");RUN(fncGo);],135,100,TOUCH);
POSN(+194,+0); KEY(keyFlrG, [LOAD(vReqd, 0);TEXT(txtCurFlr, "G");RUN(fncGo);],135,100,TOUCH);
POSN(+193,+0); KEY(keyFlrP1,[LOAD(vReqd,-1);TEXT(txtCurFlr,"P1");RUN(fncGo);],135,100,TOUCH);
```

```
POSN(+192,+0); KEY(keyFlrB2,[LOAD(vReqd,-2);TEXT(txtCurFlr,"B2");RUN(fncGo);],135,100,TOUCH);

POSN(275,430);TEXT(txtCurFlrLbl,"CURRENT FLOOR",stTxt48W);
POSN(540,+0); TEXT(txtCurFlr,"G",stTxt48W);
LOOP(lpLiftMain,FOREVER){RUN(fncDemoUpdate);}
}

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(112,240);IMG(imgTri,ptrLiftArr,86,200,stGenImg);HIDE(imgTri);
POSN(373,+0);IMG(img10s,ptrLiftTens,172,240,stGenImg);
POSN(620,+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(400,239);IMG(imgDC,libImgPMTD,640,480,stGenImg);}
PAGE(pgOpen,stLiftPg){POSN(400,239);IMG(imgDO,libImgDoors,640,480,stGenImg);}
PAGE(pgBlnk,stLiftPg){}

SHOW(pgLiftMain);
```