Noritake itron GU-U100 Code Library

Document Number:	E-M-0110-00
Issue Date:	05/18/2012

Noritake Co., Inc. **Electronics Division Headquarter** 2635 Clearbrook Drive Arlington Heights, IL 60005 Toll free: (800) 779 - 5846 Phone: (847) 439 - 9020 support.ele@noritake.com www.noritake-elec.com	**East Coast** **New Jersey Branch** 15-22 Fair Lawn Ave. Fair Lawn, NJ 07410 Toll free: (888) 296 - 3423 Phone: (201) 475 - 5200 Fax: (201) 796 - 2269	**Midwest, Canada, and Mexico** Chicago Branch 2635 Clearbrook Dr. Arlington Heights, IL 60005 Toll free: (800) 779 - 5846 Phone: (847) 439 - 9020 Fax: (847) 593 - 2285	**West Coast** **Los Angeles Branch** 21081 S. Western Ave. Ste 180 Torrance, CA 90501 Toll free: (888) 795 - 3423 Phone: (310) 320 - 1700 Fax: (310) 320 - 2900

Noritake Co., Inc., 2635 Clearbrook Drive, Arlington Heights, IL 60005, USA.

Contents

bool numberString(char buf[10], unsigned long number, uint8_t max, char fill);
bool print(int number, uint8_t max, char fill);
bool print(long number, uint8_t max, char fill);
bool print(unsigned long number, uint8_t max, char fill);
bool print(unsigned number, uint8_t max, char fill);

Graphics

void clearScreen();

Inverting Colors

void invertOn()
void invertOff();

void setDot(uint8_t x, uint8_t y, bool on);
void drawCircle(uint8_t cx, uint8_t cy, uint8_t radius, bool on);

Drawing Images

void drawImage(const uint8_t *data, uint8_t x, uint8_t y, uint8_t width, uint8_t height, uint8_t whole_width);
void drawImage(const uint8_t *data, uint8_t x, uint8_t y, uint8_t width, uint8_t height);

void drawLine(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, bool on);
void drawRect(uint8_t x1, uint8_t y1, uint8_t width, uint8_t height, bool on);
void fillRect(uint8_t x1, uint8_t y1, uint8_t width, uint8_t height, bool on);

About

The **GU-U100** is a graphic vacuum fluorescent display module meant to replace graphic LCDs based on the KS0108 chipset.

On top of supporting all of the common KS0108 commands, GU-U100 adds the unique features of screen brightness and power save mode.

Replacing LCDs

Code written for KS0108 can be used to control the GU-U100.

Depending on the LCD being replaced, slight modifications may be necessary.

The direction of **chip select (CS)** or **slave select (SS)** may be different. The GU-U100's parallel interface chip select is active high, meaning the device accepts commands only when CS is 1. Some LCDs expect chip select to be active low, meaning the device is active only when CS is 0.

The number of display segment driver chips may be different. If the code was written for more chips than is available on the GU-U100, only the left-most portions of the expected screen image will be visible.

Hardware Operation

Display Segment Driver Chips

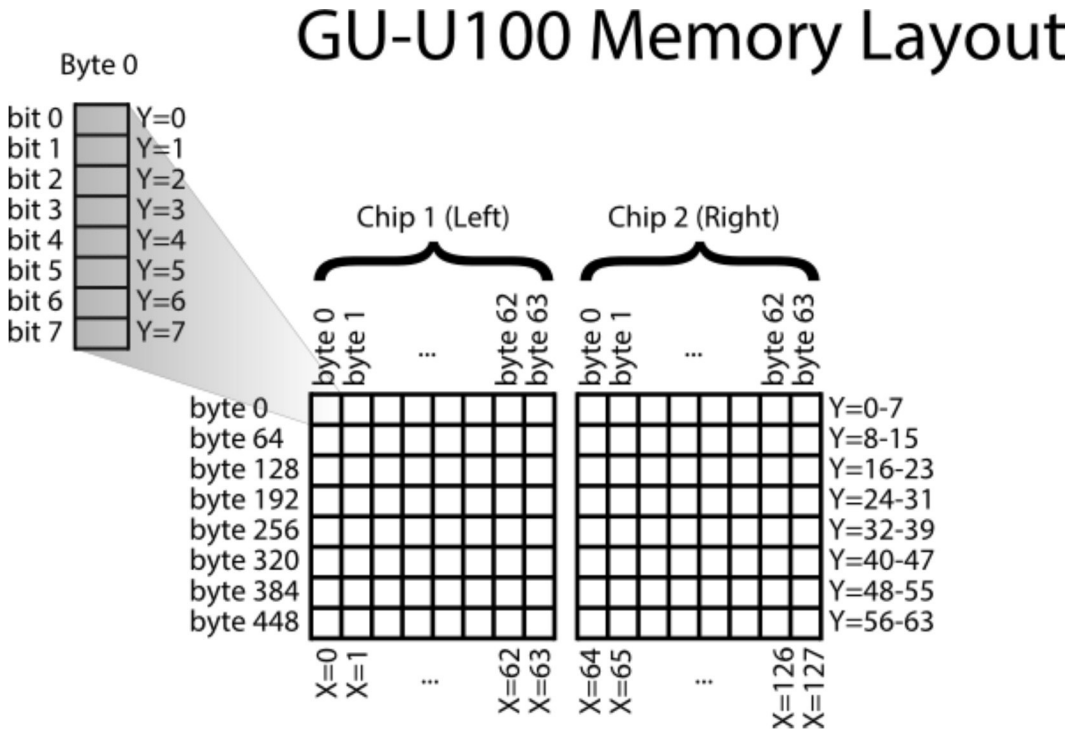
> This document assumes the device is oriented so that the longer dimension is horizontal since this is the most common use. The module specification is written from the opposite perspective (the longer dimension is vertical).

The display is divided into 64×64-dot segments arranged horizontally. Each of these segments is controlled by a separate chip. GU128x64E-U100 has two chips.

These chips do not communicate information such as the display memory or current x and y position. These must be set specifically for each chip. However, screen brightness and power save mode affect the whole module and are shared.

To access the left side of the screen, you must send commands to the first chip. The right side of the screen is accessed with the second chip. Both chips cannot be accessed simultaneously.

Memory Layout



Each controller has a 64×64-dot memory buffer representing half of the image on the screen.

- Each dot is represented by one bit in memory
- Memory is accessed by bytes (8 bits)
- Each byte represents 8 vertical dots
- Bytes are laid out horizontally
- Each page (64 bytes) represents one row: 64×8 dots
- The next page (starting from byte 64) represents the next vertical 8 dots

The address and bit value are calculated:

```
page = y/8
bit = 1 << y%8
addr = page*64 + x
```

Reading and Writing

Each chip keeps track of the current X and Y values (called the cursor).

After every memory read or write is executed, the X counter increments. After X is 63, the counter wraps back around to 0. The Y counter is not changed.

The library increments and wraps the value of Y after wrapping in the X direction.

Library Setup

Installation

This library is written for Atmel AVR microcontrollers.

Demo projects use AVR Studio 4.

1. Download the code library.
2. Unzip the library file to your work area.
3. Extract the demo folder into the folder for the GU-U100 code library.
4. Set the configuration options in *config.h* included with the code library. See the *Configuration* section.
5. Open the *Demo.aps* project file with AVR Studio 4.

Configuration

The library is configured by setting preprocessor values in the *config.h* file in the *src* directory of the library.

GU-U100 offers parallel and serial interfaces. The interface is controlled by `NORITAKE_INTERFACE`.

Consult the module specification for setting the jumpers to select the interface.

- 0 selects parallel (module default)
- 1 selects one of the serial interfaces.
  `NORITAKE_SERIAL` must be set to the correct serial interface type:
  - `1` - Type 1: *CU-UW*
  - `2` - Type 2: *SPI*
  - `3` - Type 3: *Signal Separate*

The library must be told which pins connect the host to the module. Each signal has a preprocessor variable ending with `_PORT` and `_PIN`.

E.g. for the chip select 1 pin:

```
#define CS1_PIN     3
#define CS1_PORT    PORTA
```

Parallel

Parallel is the fastest interface but uses the most pins. Each signal is controlled by a separate pin.

Code that was written for LCDs will use the parallel interface. Minor modifications may be necessary as discussed in the *Replacing LCDs* section.

```
#define DATA_PORT   PORTD
#define RS_PIN      0
#define RS_PORT     PORTA
#define RW_PIN      1
#define RW_PORT     PORTA
#define E_PIN       2
#define E_PORT      PORTA
#define CS1_PIN     3
#define CS1_PORT    PORTA
#define CS2_PIN     4
#define CS2_PORT    PORTA
#define RESET_PIN   5
#define RESET_PORT  PORTA
```

- `DATA` - data bits 0 - 7 (must be on pins 0 - 7 of given port)
- `RS` - command (0) / data (1) signal
- `RW` - write (0) / read (1) signal
- `E` - enable signal
- `CS1` - chip select 1 (left side)
- `CS2` - chip select 2 (right side)
- $\overline{\texttt{RST}}$ - reset module, active low

Serial Interface Type 1: CU-UW

This interface uses the minimum number of pins, but requires up to two bytes to be sent per data item or command. Data is input and output over the same line.

```
#define SIO_PIN     4
#define SIO_PORT    PORTG
#define CS_PIN      7
#define CS_PORT     PORTB
#define SCK_PIN     3
#define SCK_PORT    PORTG
#define RESET_PIN   0
#define RESET_PORT  PORTA
```

SI/SO, $\overline{\text{CS}}$, SCK, and $\overline{\text{RST}}$ lines are used.

The $\overline{\text{CS}}$ (chip select) here is for the entire module. It is not the same as $\overline{\text{CS1}}$ or $\overline{\text{CS2}}$.

A status byte must be sent when $\overline{\text{CS}}$ is lowered. This status byte sets the state of $\overline{\text{CS1}}$, $\overline{\text{CS2}}$, RW, and RS. Commands and data may be sent continuously without re-sending this status byte until $\overline{\text{CS}}$ is raised or the status of $\overline{\text{CS1}}$, $\overline{\text{CS2}}$, RW, or RS needs to change. After this, commands and data bytes are sent or received.

- `SIO` - I/O pin
- $\overline{\text{CS}}$ - module chip select (not to be confused with $\overline{\text{CS1}}$ and $\overline{\text{CS2}}$)
- `SCK` - synchronous serial clock
- $\overline{\text{RST}}$ - reset module, active low

Serial Interface Type 2: SPI

This interface allows you to use the module on an SPI bus as two slave devices selected by $\overline{\text{CS1}}$ and $\overline{\text{CS2}}$. Data is sent to the module on a separate line from data received from the module.

```
#define SO_PIN      4
#define SO_PORT     PORTG
#define SCK_PIN     3
#define SCK_PORT    PORTG
#define RESET_PIN   0
#define RESET_PORT  PORTA
#define CS2_PIN     1
#define CS2_PORT    PORTA
#define CS1_PIN     2
#define CS1_PORT    PORTA
#define SI_PIN      3
#define SI_PORT     PORTA
```

SO, SCK, $\overline{\text{RST}}$, $\overline{\text{CS1}}$, $\overline{\text{CS2}}$, and SI lines are used.

A status byte similar to the Type 1 interface must be sent to change RW and RS.

- `SO` - output (data/status) sent from the module to the host
- `SCK` - synchronous serial clock
- $\overline{\text{RST}}$ - reset module, active low
- $\overline{\text{CS1}}$ - chip select 1 (left side), active low
- $\overline{\text{CS2}}$ - chip select 2 (right side), active low
- `SI` - input (data/commands) sent to the module from the host

Serial Interface Type 3: Signal Separate

This has the potential to be the fastest serial interface since no status byte ever need be sent.

```
#define RS_PIN      7
#define RS_PORT     PORTB
#define SCK_PIN     3
#define SCK_PORT    PORTG
#define RESET_PIN   0
#define RESET_PORT  PORTA
#define CS2_PIN     1
#define CS2_PORT    PORTA
#define CS1_PIN     2
#define CS1_PORT    PORTA
#define SI_PIN      3
#define SI_PORT     PORTA
```

Since there is no pin to output from the module, many methods may be restricted in use since they rely on reading the display memory.

This interface is primarily useful for simple or pre-rendered high-speed animation with fewer pins than the parallel interface.

RS, SCK, $\overline{\text{RST}}$, CS1, CS2, and SI lines are used.

- `RS` - command (0) / data (1) signal
- `SCK` - synchronous serial clock
- $\overline{\text{RST}}$ - reset module, active low
- `CS1` - chip select 1 (left side), active high
- `CS2` - chip select 2 (left side), active high

- `SI` - input (data/commands) sent to the module from the host

Method Reference

Convenience Methods

uint8_t align(uint8_t y);

Align a y value to the next highest 8-vertical-dot block (i.e. values that are not divisible by 8 are rounded up).

y	y value to align

Returns: Aligned y value.

uint8_t clip(uint8_t y);

Align a y value to the previous 8-vertical-dot block (i.e. values that are not divisible by 8 are rounded down).

y	y value to clip

Returns: Clipped y value.

uint8_t fontAdvance();

Get the font width including the gutter space. The cursor will be moved horizontally by this amount when working with text-based methods.

Returns: Advance width of the font: `fontWidth + fontGutter`

Primitives

void init();

Initialize the module.

This must be called after `reset()`.

Module State:

- x = 0
- y = 0
- display = on
- power save mode = off
- brightness = 100%
- font = NULL
- fontWidth = 0
- fontHeight = 0
- fontGutter = 1

void reset();

Reset the module.

After a reset, call `init()` before any other method.

void command(uint8_t bits, bool chip);

Send a command to the module. The command's bit patterns are detailed in the module's specification.

The library does not attempt to understand commands. If a setting on the module is changed, the member variables of the library should be updated to reflect those changes.

Use caution when sending commands.

bits	command to send
chip	chip to send the command to

void setCursor(uint8_t x, uint8_t y);

Set the cursor to (x, y).

If (x, y) is beyond the limits of the screen, the request is ignored.

If the cursor is already at the given position, no command is sent to the module.

The cursor in the module only accepts y values aligned to 8-vertical-dot blocks. If the y is not divisible by 8, it is rounded down and the when used with `writedata()` and `readData()`.

Other methods compensate for the misalignment.

x	x coordinate: 0 ≤ x < 128
y	y coordinate: 0 ≤ y < 64

void writeData(uint8_t data);

Write to the display memory at the cursor position. If the cursor y was not divisible by 8, then it is rounded down to the previous 8-vertical-dot block and all 8 vertical dots will be overwritten.

This increments the x counter and causes wrapping. If x was 127, then x will be 0 and y will be y+8. If y is greater than 64, y wraps to 0.

data	data to write to the display memory

uint8_t readData();

Read from the display memory at the cursor position. If the cursor y was not divisible by 8, then it is rounded down to the previous 8-vertical-dot block and all 8 vertical dots will be read.

Serial interface type 3 always returns 0.

This increments the x counter and causes wrapping. If x was 127, then x will be 0 and y will be y+8. If y is greater than 64, y wraps to 0.

Returns: Data read from the display memory.

uint8_t peek(uint8_t x, uint8_t y);

Read from the display memory at the given position. `setCursor(x, y)` is called before and after `readData()`.

Serial interface type 3 always returns 0.

Returns: The data read from the display memory at the given position.

uint8_t readStatus(bool chip);

Read the status of the chip.

Serial interface type 3 always returns 0.

Returns: Chip status as a combination of:

- 0x80 - Busy
- 0x20 - Chip Off
- 0x10 - Resetting

void setScreenBrightness(uint8_t percent);

Set screen brightness.

This will bring the module out of power save mode.

percent	brightness value: 13% ≤ percent < 100%. Percentages are rounded up to the next selectable value: • 100% • 87.5% • 75% • 62.5% • 50% • 37.5% • 25% • 12.5%

void displayOff(bool powerSave);

Turn the display off.

Power save mode consumes less power than both turning all dots off and turning the display off but requires more time to turn back on.

powerSave	`true` enables power save mode; `false` disables power save mode

void displayOn();

Turn the display on after it has been turned off. This will also disable power save mode.

Character-Based Movements

void back();

Move the cursor back by one character width in the current font. If the cursor is within one character width of the start of a line, the cursor moves to the end

See `fontAdvance()`.

of the previous line. If the cursor was within one character width of (0, 0), it is moved to (0, 0).

This can also be used by printing `'\b'`.

void carriageReturn();

Move the cursor to the beginning of the current line.

This can also be used by printing `'\r'`.

void crlf();

Move the cursor to the beginning of the next line as though `carriageReturn()` and `lineFeed()` were called.

This can also be used by printing `"\r\n"`.

void forward();

Move the cursor forward by one character width in the current font. If the cursor is within one character width of the right edge of the screen, the cursor is moved to the beginning of the next line. If the cursor is at the end of the display, the cursor is moved to (0, 0).

See `fontAdvance()`.

This can also be used by printing `'\t'`.

void home();

Move the cursor to (0, 0).

This can also be used by printing `'\x0b'`.

void lineFeed();

Move the cursor to the next line. If on the last line of the display, the cursor is moved to (0, 0).

This can also be used by printing `'\n'`.

Printing

void setFont(const uint8_t *data, uint8_t width, uint8_t height);

This sets the current font, its width, and its height.

Each character is a width×height bitmap. The bitmaps for each character defined must come one after another (aligned to bytes) in Flash ROM (program ROM). Characters from `0x20` to `0xff` may be defined.

See the `drawImage()`.

data	data for characters in bitmap format in Flash ROM (program ROM)
width	width of each character not including gutter space
height	height of each character

Printing Characters and Strings

There are many variations of the print method. Variations that end with `_p` print from the Flash ROM (program ROM) of the host. Variations that end with `ln` print and then go to the next line as if `carriageReturn()` and `lineFeed()` had been called.

The background the size of fontAdvance()×fontHeight is cleared before drawing the character. The bottom dots of the bottom group of 8 vertical dots are preserved if the font height is not a multiple of 8. E.g. if the font is defined to be 10 dots high, the top 10 dots are cleared, and the bottom 6 are left untouched.

When used with serial interface type 3, all 8-vertical-dot groups that the character touches will be cleared.

When the cursor is within one character width of the end of the screen, text is wrapped to the next line as though `crlf()` was called.

See `fontAdvance()`.

Characters below `0x20` are control characters. Any control character that is not defined is ignored.

See *Character-Based Movements* and `clearScreen()`.

```
void print(char c);
void print(const char *buffer, size_t size);
void print(const char *str);
void print_p(const char *buffer, size_t size);
void print_p(const char *str);
void println(char c);
void println(const char *buffer, size_t size);
void println(const char *str);
void println_p(const char *buffer, size_t size);
void println_p(const char *str);
```

Printing Numbers

Print a number `max` characters long.

The `fill` character is inserted on the left if the number has less than `max` digits.

- `' '` (Space) right-aligns
- `'0'` prefixes the number with zeroes
- `0` prints only the necessary digits

```
bool numberString(char buf[10], unsigned long number, uint8_t max, char
fill);
bool print(int number, uint8_t max, char fill);
bool print(long number, uint8_t max, char fill);
bool print(unsigned long number, uint8_t max, char fill);
bool print(unsigned number, uint8_t max, char fill);
```

Returns: false and does not print any characters if:

- `max` is > 10
- `number` requires more than `max` digits

Graphics

void clearScreen();

Clear the screen and move the cursor to (0, 0).

This can also be used by printing '\x0c'.

Inverting Colors

```
void invertOn()
void invertOff();
```

Invert the colors of drawing methods.

Only dots written after this call are affected. The existing screen image does not change.

void setDot(uint8_t x, uint8_t y, bool on);

Set the dot at (x, y).

If (x, y) is beyond the limits of the screen, the request is ignored.

x	x coordinate: 0 ≤ x < 128
y	y coordinate: 0 ≤ y < 64

This should not be used from serial interface type 3. Setting multiple dots in the same 8-vertical-dot group will overwrite existing dots.

void drawCircle(uint8_t cx, uint8_t cy, uint8_t radius, bool on);

Draw a circle centered at (cx, cy) with the given radius.

Dots beyond the limits of the screen are ignored.

This should not be used from serial interface type 3.

cx	x coordinate: 0 ≤ cx < 128
cy	y coordinate: 0 ≤ cy < 64
radius	radius of the circle: 0 ≤ cx-radius < 128 0 ≤ cx+radius < 128 0 ≤ cy-radius < 64 0 ≤ cy+radius < 64
on	`true` lights dots; `false` turns dots of the circle off

Drawing Images

Draw an image from (x, y) to (x+width, y+height). `data` is a pointer to Flash ROM (program ROM) image data. The `width` must be less than or equal to the image width in memory, `whole_width`. The height may be less than the total image height in memory.

Memory is laid out as in *Memory Layout* in *Hardware Operation*; however, bitmaps are not limited to 64×64-dot blocks.

Dots beyond the limits of the screen are ignored.

Dots in the same 8-vertical-dot group but not inside the specified rectangle are preserved when the image is drawn so that its y values are not divisible by 8.

When used with serial interface type 3, all 8-vertical-dot groups that the images

```
void drawImage(const uint8_t *data, uint8_t x, uint8_t y, uint8_t width,
uint8_t height, uint8_t whole_width);
void drawImage(const uint8_t *data, uint8_t x, uint8_t y, uint8_t width,
uint8_t height);
```

touches will be cleared.

data	Image data in the same format as display memory
x	x coordinate: 0 ≤ x < 128
y	y coordinate: 0 ≤ y < 64
width	width to draw to screen: 0 ≤ x+width ≤ whole_width ≤ 128
height	height to draw to screen: 0 ≤ y+height ≤ 64
whole_width	the width of the bitmap in memory

void drawLine(uint8_t x1, uint8_t y1, uint8_t x2, uint8_t y2, bool on);

Draw a line from (x1, y1) to (x2, y2).

Dots beyond the limits of the screen are ignored.

This should not be used from serial interface type 3.

x1	start x coordinate: 0 ≤ x1 < 128
y1	start y coordinate: 0 ≤ y1 < 64
x2	end x coordinate: 0 ≤ x2 < 128
y2	end y coordinate: 0 ≤ y2 < 64
on	`true` lights dots; `false` turns dots of the line off

void drawRect(uint8_t x1, uint8_t y1, uint8_t width, uint8_t height, bool on);

Draw the outline of a rectangle from (x1, y1) to (x2, y2).

Dots beyond the limits of the screen are ignored.

This should not be used from serial interface type 3.

x	x coordinate: 0 ≤ x < 128
y	y coordinate: 0 ≤ y < 64
width	width: 0 ≤ x+width ≤ 128
height	height: 0 ≤ y+height ≤ 64
on	`true` lights dots; `false` turns dots of the rectangle off

void fillRect(uint8_t x1, uint8_t y1, uint8_t width, uint8_t height, bool on);

Fill the rectangle from (x1, y1) to (x2, y2).

Dots beyond the limits of the screen are ignored.

This should not be used from serial interface type 3 unless the rectangle begins and ends on vertical dots that are divisible by 8.

x	x coordinate: 0 ≤ x < 128
y	y coordinate: 0 ≤ y < 64
width	width: 0 ≤ x+width < 128
height	height: 0 ≤ y+height < 64
on	`true` lights dots; `false` turns dots of the rectangle off