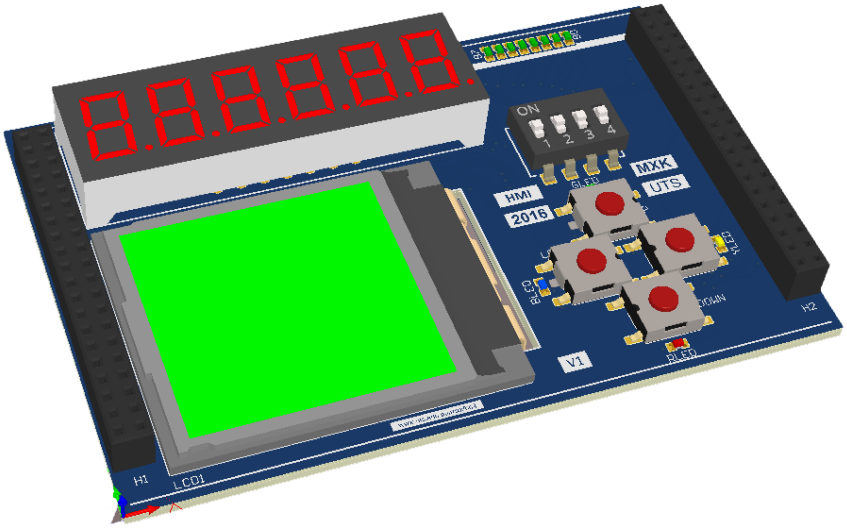



# MXK HMI User Guide

The MXK HMI is a module that provides the primary means of input and output to an assembled MXK system.

**Document Author:**
David Ledger

**Subjects:**
Introduction to Digital Systems
Advanced Digital Systems
Mechatronics 1
Mechatronics 2
DADS

Published: 21/11/2016
University of Technology, Sydney

# Contents

Nomenclature ..... 2
Features ..... 3
Interface ..... 3
Pinout ..... 3
Parts ..... 4
Functional Blocks ..... 5
MXK Serial Bus ..... 5
MXK Parallel Bus ..... 5
LCD ..... 6
Seven Segment Display ..... 7
Bar Graph ..... 7
LED's ..... 7
1.1 Buttons and DIP Switches ..... 10
Software Libraries ..... 11
HMI LED Libraries ..... 11
1.1 HMI LCD Libraries ..... 13

**No table of figures entries found.**

## ~~Glossary of Terms~~Nomenclature

**USB** (Universal Serial Bus) – A standard interface available on almost all computers.
**CAN** (Controller Area Network) – A robust and standard communications interface that is optimised for control and communication between sensors.
**SSI** (Synchronous Serial Interface) – An interface common with sensors, encoders and other industrial equipment.
**SPI** (Serial Peripheral Interface) – A serial communications protocol very common with microcontrollers, sensors, memory and other equipment.
**I2C** (Inter-Integrated Circuit) – A minimal serial interface with only two wires.
**DC** (Direct Current) – Electrical current that flows in one direction.
**PCB** (Printed Circuit Board) – An electrical circuit board.
**RoHS** (Restriction of Hazardous Substances) – In electronics, this often refers to lead free products.
**IC** (Integrated Circuit) – An electronics component that performs a specific function.
**SISO** (Single Input Single Output) – A type of controller that uses a sensor for feedback to control a device.
**IO** (Input Output) – Used to describe a port/pin that can function as an input or an output.
**UART** (Universal Asynchronous Receiver Transmitter) – A serial interface that in its most basic form it is a two-wire interface.
**ADC** (Analog to Digital Convertor) – A device that converts an analog signal (a voltage) to a digital value.
**IC** (Integrated Circuit) – A compact device that serves a specific function within a discrete package.

The following outlines the interface for the Human Machine Interface(HMI) elements (LEDS, Screens, Buttons). It has a 75x50mm footprint, a standard MXK PCB 2.

## Features

The following list outlines the features of the MXK HMI:

- 128 x 128-pixel Colour LCD.
- Selectable parallel or Serial Interface for system.
- 16 Possible Addresses for multiple display solutions.
- C Graphics and system libraries to manage display's.
- 1x 8 LED bargraph.
- 4x DIP switches for configuration mode.
- 4x Directional Buttons (Up, Down, Left and Right).
- 6x Seven Segment Display with Decimal Point (DP).

## Interface

The MXK HMI has two connectors which function differently. These differences enable the device to be driven by either the FPGA or the Microcontroller, or both.

For more information see "MXK Interface User Guide".

## Pinout

The following represents the pin allocations for the connectors on the MXK HMI:

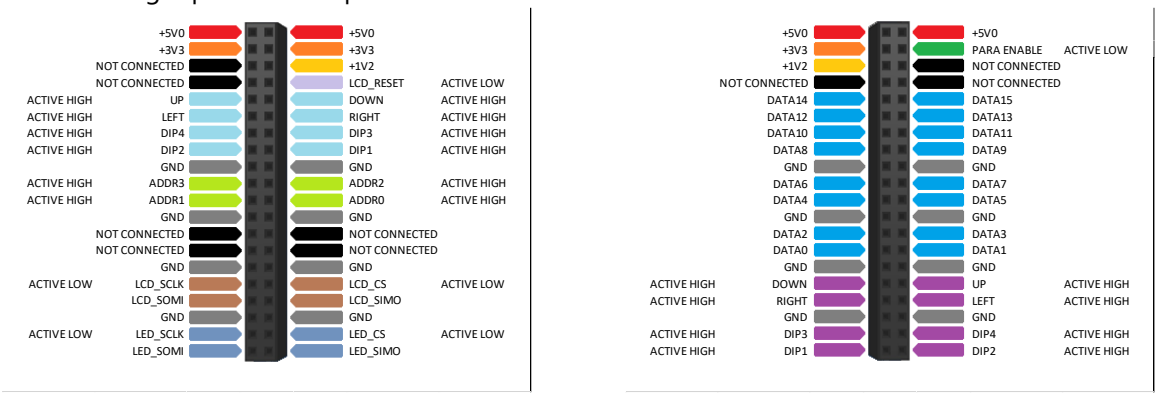



**Figure 1 Left, Serial Drive Header. Right, Parallel Drive Header.**

## Parts

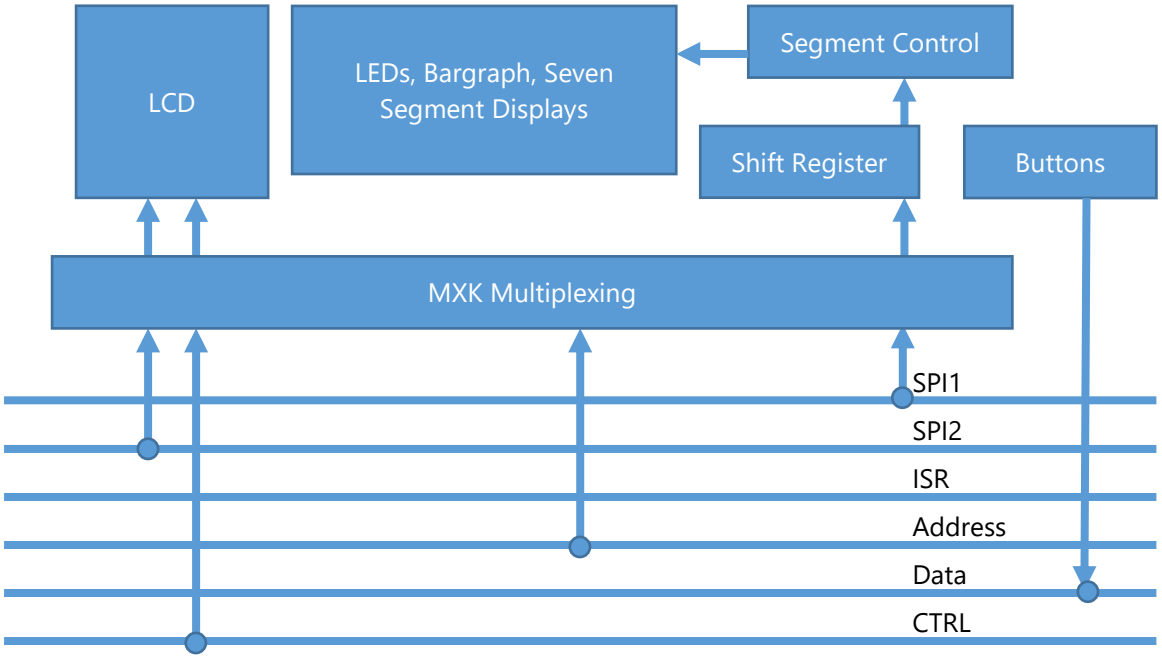
The following outlines the key parts in the module:

| Part | Part Number | Details |
|---|---|---|
| **Seven Segment Display** | KEM-3661-BSR | A 6 Digit Red Seven segment display with a decimal point. |
| **TFT LCD Screen** | MTF0144SN-08 | A 128x128-px colour TFT LCD screen with 9 bit SPI interface. |
| **Tactile Switches** | ULO-MSS497C | A low profile tactile switch without keycap. |
| **DIP Switch** | ULO-MDS428A | DIP switch with 4 switches. |

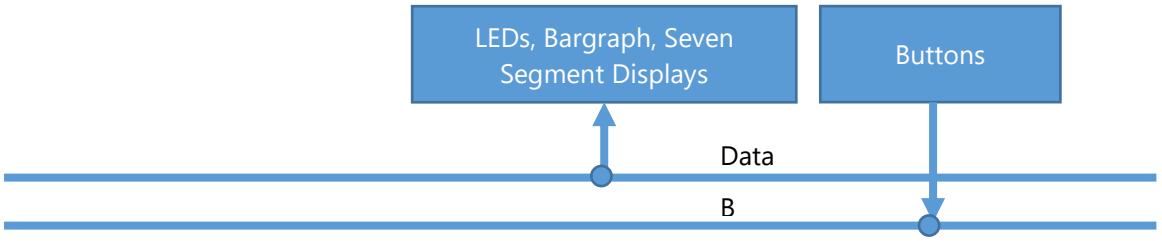
## Functional Blocks

The following is a high-level block diagrams of all the sections of the MXK HMI Module.

### MXK Serial Bus




### MXK Parallel Bus




The optimal setup for this module is driven in serial format via the FPGA. This enables the FPGA to be used as a repeater(literally repeating over and over, x times a second) for the LED rendering commands, the FPGA has no issue fowarding commands from the microcontroller (for the LCD) through the bus. This setup enables constant refreshrate on the bargraphs etc and also enables the LCD to be updated using the LCD libraries via the microcontroller module.

## LCD

The LCD on the MXK HMI is a colour 128x128-px TFT LCD display. Information can be rendered to the display via 9-bit SPI commands. This is a bit pattern cannot be generated via the inbuilt MSSP SPI module (it only does 8 bits). This means that to interface with the device it must be bit-banged or a combination of SPI transactions and bit banging must be performed. Libraries are provided to do either of the above. The LCD works with a single buffer which is equivalent to the following in C:

```
1.  #define HEIGHT 128
2.  #define WIDTH 128
3.  Pixel Buffer[HEIGHT][WIDTH];
```

The above in terms of commands can be treated as a single dimension array:

```
3.  Pixel Buffer[HEIGHT*WIDTH];
```

For rendering to be performed most LCD require a rendering window to be set, this is the rectangle that the buffer will be unloaded into. The window for a full screen refresh is 128x128, with coordinates 0,0. For a command to be performed
The bytes are transmitted using in the following order, where row are rendered one after another:

```
1.  UINT8 x, y;
2.
3.  for (y = 0; y < HEIGHT; y++)
4.      for (x = 0; x < WIDTH; x++)
5.          OUTPUT (Pixel (x, y));
```

Each pixel for the LCD consist a 16-bit colour. There are two common 16-bit colour formats:

5 Red Bits, 6 Green Bits, 5 Blue Bits
5 Red Bits, 5 Green Bits, 5 Blue Bits, 1 Unused Bit

The LCD on the MXK HMI uses the 5, 6, 5 format.

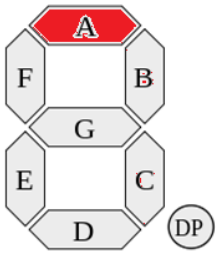
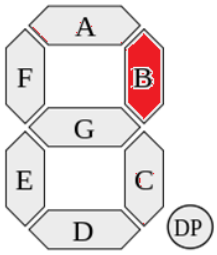
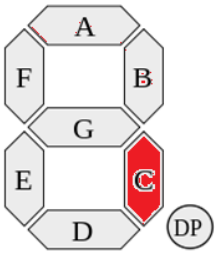
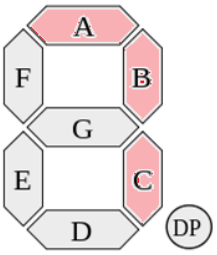
$$C = N\frac{2^{n_2}}{2^{n_1}}$$

Rounding should be performed on the above result.

## Seven Segment Display

A seven-segment display is a group of LED arranged into a figure eight pattern (some have decimal points). Each of the LED's in this pattern are diffused into the shape of a segment. These devices can be driven through various means. The HMI board provides two means, both methods rely on strobing a section of the desired display so that it is on for a portion of strobing period. The strobing must be sufficiently fast to avoid visible flicker but slow enough to enable the LED to settle to a steady state before moving to the next segment.

An example for a strobing sequence is seen below:

| T=1 | T=2 | T=3 | Average |
|---|---|---|---|
|  |  |  |  |



The HMI module has 6 segments, each with the ability to output a decimal point. This means the range of output is:

$$0.00001 \leq Output \leq 100000$$

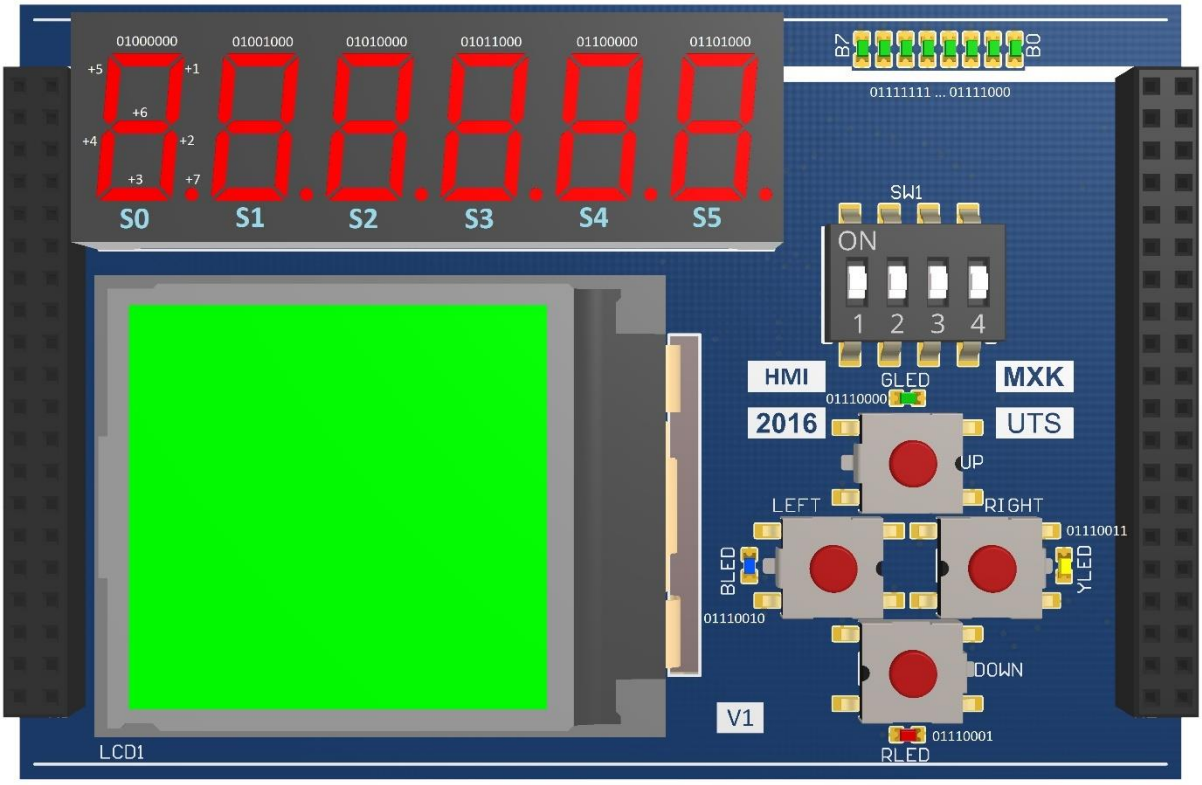
Anything outside of this range needs to be scaled to the next available SI unit value (g to kg).

## Bar Graph

The bargraph on the MXK HMI is simply 8 LED soldered next to each other. They can be driven in an identical manner to the seven-segment display or directly driven via parallel drive header.

## LED's

The LED's are located around the buttons; they can be used to prompt user input on buttons. They are driven identically to the Bar Graph.



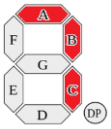

To use elements of the HMI you need to access microcontrollers SPI port, or use the FPGA to manually output a stream of bits (one after the other). Alternatively, the system can be driven through the parallel drive header.

| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|
| NC | **Enable**<br>This enables the address specified. | **GROUP ADDRESS**:<br>**0b000**: S0 is the first display.<br>**0b001**: S1 is the second display.<br>**0b010**: S2 is the third display.<br>etc…. | | | **SEGMENT ADDRESS**:<br>**0b000**: Is the segment A<br>**0b001**: Is the segment B<br>**0b010**: Is the segment C<br>**0b011**: Is the segment D<br>**0b100**: Is the segment E<br>**0b101**: Is the segment F<br>**0b110**: Is the segment G<br>**0b111**: Is the segment DP | | |

In the example code at the bottom of this document a single character is defined by a constant, this constant is found from the following table.

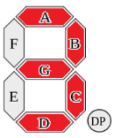
| Bit | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|
| **Segment** | DP | G | F | E | D | C | B | A |

The constants used in the program are found using the following technique:



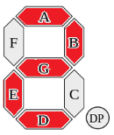

The number seven (line 4 in code) is:

| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | HEX |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0x07 |




The number three (line 5 in code) is:

| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | HEX |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0x4F |




The number two (line 6 in code) is:

| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | HEX |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0x5B |

An example of C code for rendering a single character would be:

```
1.  #include <stdio.h>
2.  typedef unsigned char UINT8;
3.  typedef unsigned int UINT;
4.  const UINT8 SEVEN = 0b00000111;
5.  const UINT8 THREE = 0b01001111;
6.  const UINT8 TWO = 0b01011011;
7.  const UINT8 STOP = 0;
8.
9.  void OUTPUT(UINT pOutput)
10. {
11.     //This should output on the SPI channel
12. }
13. //The following program will render 372
14. int main()
15. {
16.     char str[] = {SEVEN, THREE, TWO, STOP};
17.     UINT8 a, cur, out, segset;
18.     UINT8 ind = 0;
19.     cur = str[0];
20.     while (cur != STOP)
21.     {
22.         //This outputs individual segments
23.         for (a = 0; a<8; a++)
24.         {
25.             segset = (cur & 1)<<6;
26.             //0...a...7
27.             out = a | segset;
28.             OUTPUT (out);
29.             cur >>= 1;
30.         }
31.         cur = str[++ind];
32.     }
33.    return 0;
34. }
```

The following table contains a lookup table in C for all the standard letters and numbers on the seven-segment display. Some cannot be achieved on the display; approximations were made:

```
enum SevenSegmentLetters
{
    e0 = 0b00111111,
    e1 = 0b00000110,
    e2 = 0b01011011,
    e3 = 0b01001111,
    e4 = 0b01100110,
    e5 = 0b01101101,
    e6 = 0b01011111,
    e7 = 0b00000111,
    e8 = 0b01111111,
    e9 = 0b01101111,
    eA = 0b01110111,
    eB = 0b01111111,
    eC = 0b00111001,
    eD = 0b00111111,
    eE = 0b01111001,
    eF = 0b01110001,
    eG = 0b00111101,
    eH = 0b01110110,
    eI = 0b00000110,
    eJ = 0b00011110,
    eK = 0b01010111,
    eL = 0b00111000,
    eM = 0b01110110,
    eN = 0b01110110,
    eO = 0b00111111,
    eP = 0b01110011,
    eQ = 0b00111111,
    eR = 0b01110111,
    eS = 0b01101101,
    eT = 0b00110001,
    eU = 0b00111110,
    eV = 0b00111110,
    eW = 0b01111110,
    eX = 0b01110110,
    eY = 0b01100110,
    eZ = 0b01011011,
    eDot = 0b10000000
};
const char Letters [] =
{eA, eB, eC, eD, eE, eF, eG, eH, eI, eJ, eK, eL, eM, eN, eO, eP, eQ, eR, eS, eT, eU
, eV, eW, eX, eY, eZ};

const char Numbers [] =
{ e0, e1, e2, e3, e4, e5, e6, e7, e8, e9};

const char Decimal = eDot;
```

## 1.1 Buttons and DIP Switches

The inputs for the boards are provided via 4 buttons a 4x DIP switch. These can be accessed through either header in parallel. The microcontroller or FPGA end of these connections should NEVER be driven as it could result in damage to the FPGA or microcontroller. The buttons are non-inverting and when activated output logic high (1).

# Software Libraries

The HMI Module has an application library written in C to enable streamlined development of the software for user projects. This section will outline the available functions and provide examples for each.

## HMI LED Libraries

**HMI_Init**

*void HMI_Init(HMIPtr pInput)*

This function initialises the SPI channel to required settings for the seven-segment display and LCD.

```
1.  HMI Example;
2.  HMI_Init(&Example);
```

**HMI_Render**

*void HMI_Render(HMIPtr pInput)*

This function transmits a single frame for the LED's on the MXK HMI. Unless a repeater is in the system this command must be transmitted numerous times per second to ensure that the LED's are visible without flicker.

```
1.  HMI Example;
2.  HMI_Init(&Example);
3.  HMI_Render(&Example);
```

**HMI_GRender**

*void HMI_GRender(HMIPtr pInput)*

Renders a single group of the HMI, each call of this function automatically increments which group is being rendered. This function can be used when rendering must be done in-between many other tasks. It is often not suitable.

```
1.  HMI Example;
2.  HMI_Init(&Example);
3.  HMI_GRender(&Example);
```

**HMI_SetLeft, HMI_SetRight, HMI_SetUp, HMI_SetDown**

*void HMI_SetX(HMIPtr pInput, UINT8 pValue)*

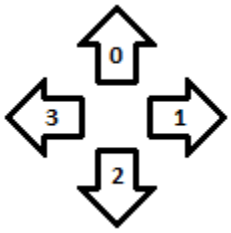
Sets the value of the directional buttons LED.

```
1.  HMI Example;
2.  HMI_Init(&Example);
3.  //Turns on the left and Up button
4.  HMI_SetLeft(&Example, ON);
5.  HMI_SetRight(&Example, OFF);
6.  HMI_SetUp(&Example, ON);
7.  HMI_SetDown(&Example, OFF);
```

**HMI_SetButton**

*void HMI_SetButton(HMIPtr pInput, UINT8 pValue)*

This sets the button indicated by the index value.
0 – Up
1 – Left
2 – Down
3 – Right




This function can be used to create a circle pattern or to set LED based on an indexing system.

```
1.  HMI Example;
2.  HMI_Init(&Example);
3.  //Turns on the left button only
4.  HMI_SetButton(&Example, 1);
5.
6.  //Turns off left, turns on down
7.  HMI_SetButton(&Example, 2);
```

**HMI_SetButtons**

*void HMI_SetButtons(HMIPtr pInput, UINT8 pValue)*

```
1.  HMI Example;
2.  HMI_Init(&Example);
3.  //Sets right and down LEDs
```

This enables the setting of a multiple buttons simultaneously. The bits correspond to the following buttons:
0 – Up
1 – Down
2 – Left
3 – Right

The function has a different indexing system to the HMI_SetButton function.

```
4.  HMI_SetButtons(&Example, 0b1010);
```

**HMI_SetBar**
*void HMI_SetBar(HMIPtr pInput, UINT8 pValue)*

Sets the bar to the value determined by the bits.

```
1.  HMI Example;
2.  HMI_Init(&Example);
3.  //Sets the first 4 LED
4.  HMI_SetButtons(&Example, 0x0F);
5.
6.  //Turns off first 4, turns on last 4
7.  HMI_SetButtons(&Example, 0xF0);
```

**HMI_SetBargraph**
*void HMI_SetBargraph(HMIPtr pInput, UINT8 pVal)*

Sets the bar to a value where all LED's are active below and at the input value (like a conventional bargraph).

```
1.  HMI Example;
2.  HMI_Init(&Example);
3.  //Sets the first 4 LED
4.  HMI_SetBargraph(&Example, 4);
```

**HMI_SetSegments**
*void HMI_SetSegments(HMIPtr pInput, str pString)*

Sets the value output on the seven-segment display from the input string. The conversion from characters to seven segment display bits is performed within this function.

```
1.  HMI Example;
2.  HMI_Init(&Example);
3.  //Sets the render buffer to "8.6180"
4.  HMI_SetSegments(&Example, "8.6180");
```

## 1.1 **HMI LCD Libraries**

**LCD_Init**
*void LCD_Init(void)*

Initialises the serial port and sets up the LCD.

This function MUST be called for the LCD to be responsive to any other commands.

```
1.  LCD_Init();
```

**LCD_Fill**
*void LCD_Fill(Colour pColour)*

Fills the screen with the colour specified in the parameter.

```
1.  LCD_Init();
2.
3.  //Fills the screen with RED
4.  LCD_Fill(RED);
```

**LCD_FillRectangle**
*void LCD_FillRectangle*
*(Colour pColour, Point pPoint, Size pSize)*

Fills a rectangle specified by the Point and Size parameters, the top left corner is specified by pPoint. The width and height of the rectangle are specified by pSize.

```
1.  LCD_Init();
2.  Point p1 = {10, 10};
3.  Point p2 = {50, 80};
4.
5.  //Fills a red rectangle
6.  LCD_FillRectangle(RED, p1, p2);
```

**LCD_DrawPixel**
*void LCD_DrawPixel (Colour pColour, Point pPoint)*

Draws a *pColour* pixel at the location specified by pPoint.

```
1.  LCD_Init();
2.  Point p1 = {10, 10};
3.
4.  //Draws a red pixel
5.  LCD_DrawPixel(RED, p1);
```

**LCD_DrawLine**
*void LCD_DrawLine*
*(Colour pColour, Point pPoint1, Point pPoint2)*

Draws a line between the two points.

```
1.  LCD_Init();
2.  Point p1 = {10, 10};
3.  Point p2 = {50, 80};
4.
5.  //Draws a red line
6.  LCD_DrawLine(RED, p1, p2);
```

**LCD_DrawField**
*void LCD_DrawField*
*(ColourPtr pField, Point pPoint, Size pSize)*

Draws a 2D colour array to the point specified.

The pSize parameter MUST be correct as this function does not perform scaling.

```
1.  LCD_Init();
2.  Size FieldSize = {20, 20};
3.  Point Location = {10, 20};
4.
5.  //Creates a multicolour field
6.  Colour Field[20][20];
7.  for (UINT8 x = 0; x < 20; x++)
8.      for (UINT8 y = 0; y < 20; y++)
9.          Field[x][y]=x+y;
10.
11. //Draws a red line
12. LCD_DrawField(&Field, Location, FieldSize);
```

**LCD_DrawRectangle**
*void LCD_DrawRectangle*
*(Colour pColour, Point pPoint1, Point pPoint2)*

```
1.  LCD_Init();
2.  Point p1 = {10, 10};
3.  Point p2 = {50, 80};
4.
5.  //Draws a red rectangle
```

| | |
|---|---|
| Draws a rectangle specified by the Point and Size parameters, the top left corner is specified by pPoint. The width and height of the rectangle are specified by pSize. | `6. LCD_FillRectangle(RED, p1, p2);` |
| **LCD_FillCircle**<br>*void LCD_DrawCircle*<br>*(Colour pColour, Point pCentre, UINT16 pRadius)*<br><br>Fills a *pColour* circle with its centre in at pCentre and radius specified by pRadius. | `1. LCD_Init();`<br>`2. Point p1 = {50, 50};`<br>`3.`<br>`4. //Fill a red rectangle with radius 25`<br>`5. LCD_FillCircle(RED, p1, 25);` |
| **LCD_DrawCircle**<br>*void LCD_FillCircle*<br>*(Colour pColour, Point pCentre, UINT16 pRadius)*<br><br>Draws a *pColour* circle with its centre in at pCentre and radius specified by pRadius. | `1. LCD_Init();`<br>`2. Point p1 = {50, 50};`<br>`3.`<br>`4. //Draw a red rectangle with radius 25`<br>`5. LCD_DrawCircle(RED, p1, 25);` |

## 1.1 **Console Libraries**

To use the LCD in a similar manner to is used in UNIX or windows terminal windows include the "Console.h" header. This will enable printf to function in the typical manner. All functions below require that the LCD has been initialised.

| | |
|---|---|
| **Console_PrintChar**<br>*void Console_PrintChar(char pInput)*<br><br>Prints a single character on the screen, the cursor is automatically incremented.<br>Some control characters are supported:<br>\n: Newline<br>\t: Tab<br>\b: Backspace<br>\f: Form feed<br>\a: Increment cursor. | Console_PrintChar('k'); |
| **Console_PrintString**<br>*void Console_PrintString(str pInput)*<br><br>Prints a C string to the console, the string must be terminated with a 0. | Console_PrintString("This is a string"); |
| **Console_ClearLine**<br>*inline void Console_ClearLine()*<br><br>Clears the current line. | Console_ClearLine(); |
| **Console_MakeField**<br>*ColourPtr Console_MakeField*<br>*(char pInput)*<br><br>Builds a render-able colour array (from forecolour and backcolour) that represents a character. | ColourPtr Array = Console_MakeField('a'); |
| **Console_SetForecolour**<br>*inline void Console_SetForecolour*<br>*(UINT16 pColour)*<br><br>Sets the text colour. | Console_SetForecolour(RED); |
| **Console_SetBackcolour**<br>*inline void Console_SetBackcolour*<br>*(UINT16 pColour)*<br><br>Sets the colour behind the text. | Console_SetBackcolour(BLUE); |
| **Console_SetForecolourRGB**<br>*inline void Console_SetForecolourRGB*<br>*(UINT8 r, UINT8 g, UINT8 b)*<br><br>Sets the text colour from separate 8 bit RGB components. | Console_SetForecolour(255, 0, 0); |
| **Console_SetBackcolourRGB**<br>*inline void Console_SetBackcolourRGB*<br>*(UINT8 r, UINT8 g, UINT8 b)* | Console_SetBackcolour(0, 0, 255); |

| | |
|---|---|
| Sets the colour behind the text from separate 8 bit RGB components. | |
| **printf**<br>*int printf ( const char * format, ... )*<br><br>See http://bit.ly/2fYeaik | See examples in link. |