PModCLS
SPI Interface with VHDL

Ergün Cömert
2011
# Table of Contents

Introduction ................................................................................................................................. 2  
SPI (Serial Peripheral Interface) ................................................................................................. 3  
PModCLS Board ............................................................................................................................ 4  
  Command to PModCLS .............................................................................................................. 6  
  Characters to PModCLS ............................................................................................................ 7  
  Custom Characters .................................................................................................................... 8  
TOP_MODULE ................................................................................................................................ 9  
  FSM_SPI Module ..................................................................................................................... 10  
  FSM_CTRL Module .................................................................................................................. 12  
  ROM_MEMORY Module ............................................................................................................. 13  
  CLOCK_DIVIDER Module ........................................................................................................ 13  
SIMULATIONS ............................................................................................................................. 14  
Timing Closure ............................................................................................................................ 19  
REFERENCES ............................................................................................................................. 20  
SOURCE CODES ......................................................................................................................... 21  
  FSM_SPI.vhd ........................................................................................................................ 21  
  FSM_CTRL.vhd ...................................................................................................................... 22  
  ROM_Memory.v ....................................................................................................................... 24  
  CLOCK_DIV.vhd .................................................................................................................... 24  
  TOP_MODULE.vhd .................................................................................................................. 25  
  LCD.ucf .................................................................................................................................. 26  
  ROM.txt ................................................................................................................................. 26
**Introduction**

BASYS2 circuit design and implementation platform from Digilent Inc. has a Xilinx Spartan 3E FPGA, switches, buttons, LEDs, seven-segment display, PS2 and VGA ports. However, it lacks of LCD display, which is required in many applications with user interface. Digilent solves this problem by providing serial and parallel interfaced external LCD with an integrated controller. Although datasheet and schematic of this LCD is present on their webpage, lack of any example code increases design time. In this document, programming the LCD with serial interface (PModCLS) using SPI communication is shown. VHDL codes are also included. The application is writing “OKAN UNIVERSITY” on LCD screen and implementing a PACMAN animation. Connection between FPGA’s JTAG-A port and PModCLS J1 connector is performed by a 6 pin header and cable connector. Final version of the design is shown in Figure 1.

![Figure 1 Connection schematic and picture of the final design](image_url)
SPI (Serial Peripheral Interface)

SPI is a synchronous serial data link between a master and multiple slave devices. Since clock is generated by master, clock requirements of slave devices are reduced unlike asynchronous communication.

An SPI device can either be a master or a slave. Master generates SS (Slave Select), MOSI (Master Output Slave Input), and SCK (Serial Clock) signals. Slave device generates MISO (Master Input Slave Output) signal (Figure 2). SS signal is an active low enable used by master to enable slave.

![SPI Master and Slave Diagram](image)

Figure 2 Serial Peripheral Interface (SPI)

In this design, master is the FPGA board, Slave is the PModCLS board. During each SPI clock cycle, full duplex data transmission occurs:

- Master sends a bit on the MOSI line; slave reads it from the same line.
- Slave sends a bit on the MISO line; master reads it from the same line.

If data is sent on the MOSI line from master to slave, previous data is sent on the MISO line from slave to master (Figure 3).

![Master to Slave Communication Diagram](image)

Figure 3 Master to slave communication figure

For example, first data is 0x1B which corresponds “ESC” character in ASCII code and second data is 0x5B which corresponds “[” character. When first data is sent to slave on the MOSI line, last data is sent back to master on the MISO line; which stored in slave memory.
When second data is sent on the MOSI line, first data 0x1B is sent back on the MISO line. These changes are shown on the Figure 4.

Figure 4 Relationships between MOSI and MISO

In this design, data is read from ROM and converted from parallel to serial with shifting operations, in order to send data to the MOSI line bit by bit. MSB (Most Significant Bit) is sent first, and then other bits are sent. Process is completed by sending the LSB (Least Significant Bit).

SCK is provided either reading or writing data synchronously. Data is changed at the falling edge of SCK, and sampled by the slave at the rising edge of SCK.

**PModCLS Board**

PModCLS board is capable of executing a variety of instructions, such as erasing specific characters, setting different display modes, scrolling, and displaying user-defined characters. These instructions are specified using escape sequences to send commands to the board’s embedded Atmel ATmega48 microcontroller, which controls all features of PModCLS board.
Connector definitions for communication methods are listed on Tables 1-3.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SS/RST</td>
<td>Slave Select</td>
</tr>
<tr>
<td>2</td>
<td>MOSI</td>
<td>Master out / Slave in Data</td>
</tr>
<tr>
<td>3</td>
<td>MISO</td>
<td>Master in / Slave out Data</td>
</tr>
<tr>
<td>4</td>
<td>SCK</td>
<td>Serial Clock</td>
</tr>
<tr>
<td>5</td>
<td>GND</td>
<td>Power supply ground</td>
</tr>
<tr>
<td>6</td>
<td>VCC</td>
<td>Power supply (3.3V)</td>
</tr>
</tbody>
</table>

Table 1 Connections for SPI communication

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SCL</td>
<td>TWI Clock</td>
</tr>
<tr>
<td>2</td>
<td>SDA</td>
<td>TWI Data</td>
</tr>
<tr>
<td>3</td>
<td>TXD</td>
<td>UART transmit Data</td>
</tr>
<tr>
<td>4</td>
<td>RXD</td>
<td>UART receive Data</td>
</tr>
<tr>
<td>5</td>
<td>GND</td>
<td>Power supply ground</td>
</tr>
<tr>
<td>6</td>
<td>VCC</td>
<td>Power supply (3.3V)</td>
</tr>
</tbody>
</table>

Table 2 Connections for UART / TWI communication

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SCL</td>
<td>TWI Clock</td>
</tr>
<tr>
<td>2</td>
<td>SDA</td>
<td>TWI Data</td>
</tr>
</tbody>
</table>

Table 3 Connections for TWI Daisy Chain

PModCLS board’s communication method is set by mode jumpers MD0, MD1, and MD2. Possible mode jumper configurations are listed in the Table 4. For Rev D boards, a missing jumper is represented by 0 and a connected jumper is represented by 1. For Rev E boards, a missing jumper is represented by 1 and a connected jumper is represented by 0.

<table>
<thead>
<tr>
<th>MD2, MD1, MD0</th>
<th>Protocol</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0, 0</td>
<td>UART</td>
<td>2400 baud</td>
</tr>
<tr>
<td>0, 0, 1</td>
<td>UART</td>
<td>4800 baud</td>
</tr>
<tr>
<td>0, 1, 0</td>
<td>UART</td>
<td>9600 baud</td>
</tr>
<tr>
<td>0, 1, 1</td>
<td>UART</td>
<td>baud rate in EEPROM</td>
</tr>
<tr>
<td>1, 0, 0</td>
<td>TWI</td>
<td>address: 0x48</td>
</tr>
<tr>
<td>1, 0, 1</td>
<td>TWI</td>
<td>address in EEPROM</td>
</tr>
<tr>
<td>1, 1, 0</td>
<td>SPI</td>
<td>specified in EEPROM</td>
</tr>
<tr>
<td>1, 1, 1</td>
<td>specified in EEPROM</td>
<td>specified in EEPROM</td>
</tr>
</tbody>
</table>

Table 4 Mode jumpers for communication methods

In this document, communication with Atmel microcontroller on the PModCLS board is established using SPI (Serial Peripheral Interface) through J1 connector.
Command to PModCLS

Sending a command to PModCLS is defined as an escape sequence in reference manual. An escape sequence is specified by first sending “ESC” character followed by a left square bracket “[” and “0”. If the command requires parameters, they are separated by semicolons “;” followed by command character for the specific command. Instruction symbols’ definitions are listed in Table 5; and instructions are listed in Table 6. Note that if there isn’t an instruction symbol before command character in Table 6, “0” character is sent before command character. For example; to send “ESC[0j”, put the following ASCII codes on MOSI: “0x1B 0x5B 0x30 0x6A”.

<table>
<thead>
<tr>
<th>Instruction Symbols</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pr&gt;</td>
<td>row number (0 – 1)</td>
</tr>
<tr>
<td>&lt;pc&gt;</td>
<td>column number (0 – 39)</td>
</tr>
<tr>
<td>&lt;pn&gt;</td>
<td>numeric parameter (decimal, hex or binary)</td>
</tr>
<tr>
<td>&lt;ps&gt;</td>
<td>decimal selection parameter</td>
</tr>
<tr>
<td>&lt;pt&gt;</td>
<td>character table selector(0 – 2 in EEPROM, 3 in RAM)</td>
</tr>
</tbody>
</table>

Table 5 Instructions symbols definitions

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pr&gt;;&lt;pc&gt;H</td>
<td>set cursor position to &lt;pr&gt; and &lt;pc&gt;</td>
</tr>
<tr>
<td>s</td>
<td>save cursor position</td>
</tr>
<tr>
<td>u</td>
<td>restore saved cursor position</td>
</tr>
<tr>
<td>j</td>
<td>clear display and home cursor</td>
</tr>
<tr>
<td>&lt;ps&gt;K</td>
<td>erase within line</td>
</tr>
<tr>
<td></td>
<td>0 = current position to end of line</td>
</tr>
<tr>
<td></td>
<td>1 = start of line to current position</td>
</tr>
<tr>
<td></td>
<td>2 = entire line</td>
</tr>
<tr>
<td>&lt;ps&gt;N</td>
<td>erase field in current line</td>
</tr>
<tr>
<td>&lt;ps&gt; = number of chars starting at current position</td>
<td></td>
</tr>
<tr>
<td>&lt;pn&gt;@</td>
<td>scroll left &lt;pn&gt; columns</td>
</tr>
<tr>
<td>&lt;pn&gt;A</td>
<td>scroll right &lt;pn&gt; columns</td>
</tr>
<tr>
<td>*</td>
<td>reset; equivalent to cycling power of PModCLS</td>
</tr>
<tr>
<td>&lt;ps&gt;e</td>
<td>enable / disable display</td>
</tr>
<tr>
<td></td>
<td>0 = display off, backlight off</td>
</tr>
<tr>
<td></td>
<td>1 = display on, backlight off</td>
</tr>
<tr>
<td></td>
<td>2 = display off, backlight on</td>
</tr>
<tr>
<td></td>
<td>3 = display on, backlight on</td>
</tr>
<tr>
<td>&lt;ps&gt;h</td>
<td>set display mode</td>
</tr>
<tr>
<td></td>
<td>0 = wrap line at 16 characters</td>
</tr>
<tr>
<td></td>
<td>1 = wrap line at 40 characters</td>
</tr>
<tr>
<td>&lt;ps&gt;c</td>
<td>set cursor mode</td>
</tr>
<tr>
<td></td>
<td>0 = cursor off</td>
</tr>
<tr>
<td></td>
<td>1 = cursor on, blink off</td>
</tr>
<tr>
<td></td>
<td>2 = cursor on, blink on</td>
</tr>
<tr>
<td>&lt;pn&gt;a</td>
<td>save TWI address in EEPROM to &lt;pn&gt;</td>
</tr>
</tbody>
</table>
**DIGITAL DESIGN LABORATORY**

<table>
<thead>
<tr>
<th>&lt;p&gt;n</th>
<th>b</th>
<th>save baud rate value in EEPROM to &lt;p&gt;n</th>
<th>&lt;p&gt;t</th>
<th>p</th>
<th>program character table into LCD</th>
<th>&lt;p&gt;v</th>
<th>l</th>
<th>load EEPROM character table to RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;p&gt;m</td>
<td>m</td>
<td>save communication mode to EEPROM</td>
<td>&lt;p&gt;n</td>
<td>n</td>
<td>save cursor mode to EEPROM</td>
<td>&lt;p&gt;v</td>
<td>o</td>
<td>save display mode to EEPROM</td>
</tr>
</tbody>
</table>

Table 6 Instructions list and tasks (escape sequences)

**Characters to PModCLS**

Characters are written to the display simply by sending their ASCII codes (Table 7) over the communication link (MOSI) with required select (SS) and clock (SCK) signals. Characters appear on display at the current location of board’s cursor. Therefore before sending any character, cursor location must be set. After the character is put on the display, the cursor is moved to right. Therefore, once the initial location character is determined, sending text characters sequentially put the text on the LCD screen.

**Table 7 ASCII character table**

<table>
<thead>
<tr>
<th>Dec</th>
<th>Bx</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Bx</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Bx</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Bx</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Bx</th>
<th>Oct</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>001</td>
<td>NULL (null)</td>
<td>002</td>
<td>011</td>
<td>040</td>
<td>ASCII</td>
<td>32</td>
<td>20</td>
<td>040</td>
<td>ASCII</td>
<td>64</td>
<td>20</td>
<td>100</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
<td>START (start of heading)</td>
<td>002</td>
<td>011</td>
<td>041</td>
<td>ASCII</td>
<td>65</td>
<td>20</td>
<td>101</td>
<td>ASCII</td>
<td>96</td>
<td>20</td>
<td>140</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>002</td>
<td>TEXT (start of text)</td>
<td>003</td>
<td>012</td>
<td>042</td>
<td>ASCII</td>
<td>66</td>
<td>20</td>
<td>102</td>
<td>ASCII</td>
<td>97</td>
<td>20</td>
<td>141</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>003</td>
<td>END (end of text)</td>
<td>004</td>
<td>013</td>
<td>043</td>
<td>ASCII</td>
<td>67</td>
<td>20</td>
<td>103</td>
<td>ASCII</td>
<td>98</td>
<td>20</td>
<td>142</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>004</td>
<td>END or TRANSMISSION</td>
<td>005</td>
<td>014</td>
<td>044</td>
<td>ASCII</td>
<td>68</td>
<td>20</td>
<td>104</td>
<td>ASCII</td>
<td>99</td>
<td>20</td>
<td>143</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>005</td>
<td>ENQ (enquiry)</td>
<td>006</td>
<td>015</td>
<td>045</td>
<td>ASCII</td>
<td>70</td>
<td>20</td>
<td>106</td>
<td>ASCII</td>
<td>100</td>
<td>20</td>
<td>146</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>006</td>
<td>ACK (acknowledge)</td>
<td>007</td>
<td>016</td>
<td>046</td>
<td>ASCII</td>
<td>71</td>
<td>20</td>
<td>107</td>
<td>ASCII</td>
<td>101</td>
<td>20</td>
<td>147</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>007</td>
<td>BEL (bell)</td>
<td>008</td>
<td>017</td>
<td>047</td>
<td>ASCII</td>
<td>72</td>
<td>20</td>
<td>108</td>
<td>ASCII</td>
<td>102</td>
<td>20</td>
<td>148</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>010</td>
<td>BS (backspace)</td>
<td>009</td>
<td>018</td>
<td>050</td>
<td>ASCII</td>
<td>73</td>
<td>20</td>
<td>111</td>
<td>ASCII</td>
<td>103</td>
<td>20</td>
<td>149</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>011</td>
<td>TAB (horizontal tab)</td>
<td>010</td>
<td>019</td>
<td>051</td>
<td>ASCII</td>
<td>74</td>
<td>20</td>
<td>112</td>
<td>ASCII</td>
<td>104</td>
<td>20</td>
<td>150</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>012</td>
<td>LF (NL line feed, new line)</td>
<td>011</td>
<td>020</td>
<td>052</td>
<td>ASCII</td>
<td>75</td>
<td>20</td>
<td>113</td>
<td>ASCII</td>
<td>105</td>
<td>20</td>
<td>151</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>013</td>
<td>VT (vertical tab)</td>
<td>012</td>
<td>021</td>
<td>053</td>
<td>ASCII</td>
<td>76</td>
<td>20</td>
<td>114</td>
<td>ASCII</td>
<td>106</td>
<td>20</td>
<td>152</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>014</td>
<td>FF (form feed, new page)</td>
<td>013</td>
<td>022</td>
<td>054</td>
<td>ASCII</td>
<td>77</td>
<td>20</td>
<td>115</td>
<td>ASCII</td>
<td>107</td>
<td>20</td>
<td>153</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>015</td>
<td>CR (carriage return)</td>
<td>014</td>
<td>023</td>
<td>055</td>
<td>ASCII</td>
<td>78</td>
<td>20</td>
<td>116</td>
<td>ASCII</td>
<td>108</td>
<td>20</td>
<td>154</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>016</td>
<td>SO (shift out)</td>
<td>015</td>
<td>024</td>
<td>056</td>
<td>ASCII</td>
<td>79</td>
<td>20</td>
<td>117</td>
<td>ASCII</td>
<td>109</td>
<td>20</td>
<td>155</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>017</td>
<td>ST (shift in)</td>
<td>016</td>
<td>025</td>
<td>057</td>
<td>ASCII</td>
<td>80</td>
<td>20</td>
<td>118</td>
<td>ASCII</td>
<td>110</td>
<td>20</td>
<td>156</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>020</td>
<td>DLE (data link escape)</td>
<td>017</td>
<td>026</td>
<td>058</td>
<td>ASCII</td>
<td>81</td>
<td>20</td>
<td>119</td>
<td>ASCII</td>
<td>111</td>
<td>20</td>
<td>157</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>021</td>
<td>DC1</td>
<td>018</td>
<td>027</td>
<td>059</td>
<td>ASCII</td>
<td>82</td>
<td>20</td>
<td>120</td>
<td>ASCII</td>
<td>112</td>
<td>20</td>
<td>158</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>022</td>
<td>DC2</td>
<td>019</td>
<td>028</td>
<td>060</td>
<td>ASCII</td>
<td>83</td>
<td>20</td>
<td>121</td>
<td>ASCII</td>
<td>113</td>
<td>20</td>
<td>159</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>023</td>
<td>DC3</td>
<td>020</td>
<td>029</td>
<td>061</td>
<td>ASCII</td>
<td>84</td>
<td>20</td>
<td>122</td>
<td>ASCII</td>
<td>114</td>
<td>20</td>
<td>160</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>024</td>
<td>DC4</td>
<td>021</td>
<td>030</td>
<td>062</td>
<td>ASCII</td>
<td>85</td>
<td>20</td>
<td>123</td>
<td>ASCII</td>
<td>115</td>
<td>20</td>
<td>161</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>025</td>
<td>NAK (negative acknowledge)</td>
<td>022</td>
<td>031</td>
<td>063</td>
<td>ASCII</td>
<td>86</td>
<td>20</td>
<td>124</td>
<td>ASCII</td>
<td>116</td>
<td>20</td>
<td>162</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>6</td>
<td>026</td>
<td>SYN (synchronous idle)</td>
<td>023</td>
<td>032</td>
<td>064</td>
<td>ASCII</td>
<td>87</td>
<td>20</td>
<td>125</td>
<td>ASCII</td>
<td>117</td>
<td>20</td>
<td>163</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>7</td>
<td>027</td>
<td>ETB (end of trans. block)</td>
<td>024</td>
<td>033</td>
<td>065</td>
<td>ASCII</td>
<td>88</td>
<td>20</td>
<td>126</td>
<td>ASCII</td>
<td>118</td>
<td>20</td>
<td>164</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>8</td>
<td>030</td>
<td>CAN (cancel)</td>
<td>025</td>
<td>034</td>
<td>066</td>
<td>ASCII</td>
<td>89</td>
<td>20</td>
<td>127</td>
<td>ASCII</td>
<td>119</td>
<td>20</td>
<td>165</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>9</td>
<td>031</td>
<td>EM (end of medium)</td>
<td>026</td>
<td>035</td>
<td>067</td>
<td>ASCII</td>
<td>90</td>
<td>20</td>
<td>128</td>
<td>ASCII</td>
<td>120</td>
<td>20</td>
<td>166</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>A</td>
<td>032</td>
<td>SUB (substitute)</td>
<td>027</td>
<td>036</td>
<td>068</td>
<td>ASCII</td>
<td>91</td>
<td>20</td>
<td>129</td>
<td>ASCII</td>
<td>121</td>
<td>20</td>
<td>167</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>B</td>
<td>033</td>
<td>ESC (escape)</td>
<td>028</td>
<td>037</td>
<td>069</td>
<td>ASCII</td>
<td>92</td>
<td>20</td>
<td>130</td>
<td>ASCII</td>
<td>122</td>
<td>20</td>
<td>168</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>C</td>
<td>034</td>
<td>FS (file separator)</td>
<td>029</td>
<td>038</td>
<td>070</td>
<td>ASCII</td>
<td>93</td>
<td>20</td>
<td>131</td>
<td>ASCII</td>
<td>123</td>
<td>20</td>
<td>169</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>D</td>
<td>035</td>
<td>GS (group separator)</td>
<td>030</td>
<td>039</td>
<td>071</td>
<td>ASCII</td>
<td>94</td>
<td>20</td>
<td>132</td>
<td>ASCII</td>
<td>124</td>
<td>20</td>
<td>170</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>E</td>
<td>036</td>
<td>RS (record separator)</td>
<td>031</td>
<td>040</td>
<td>072</td>
<td>ASCII</td>
<td>95</td>
<td>20</td>
<td>133</td>
<td>ASCII</td>
<td>125</td>
<td>20</td>
<td>171</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>F</td>
<td>037</td>
<td>US (unit separator)</td>
<td>032</td>
<td>041</td>
<td>073</td>
<td>ASCII</td>
<td>96</td>
<td>20</td>
<td>134</td>
<td>ASCII</td>
<td>126</td>
<td>20</td>
<td>172</td>
<td>ASCII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Characters to PModCLS

Characters are written to the display simply by sending their ASCII codes (Table 7) over the communication link (MOSI) with required select (SS) and clock (SCK) signals. Characters appear on display at the current location of board’s cursor. Therefore before sending any character, cursor location must be set. After the character is put on the display, the cursor is moved to right. Therefore, once the initial location character is determined, sending text characters sequentially put the text on the LCD screen.
Custom Characters

PModCLS board can display up to eight custom characters at a time, although it is capable of storing four sets of eight characters. Storage area consists of three character tables in EEPROM and one table loaded into the LCD’s RAM. To create a new custom character, send the command “\texttt{ESC[<pn>};...;<pn>;<ps>d}”, where \texttt{<pn>} is a numeric parameter that describes a row in the character and \texttt{<ps>} is a decimal selection parameter (0 through 7.) A custom character definition contains eight rows, so the escape sequence to define one must have eight \texttt{<pn>} values separated by semicolons followed by \texttt{<ps>} value that specifies the character being defined.

The procedure to define a custom character is as follows. First draw the pattern of the character. Then, determine the numerical value of each row in the character. Note that the left most segment is the most significant bit. Record each row, from top to bottom, and record which character in the table to save it to. Each row of character contains six pixels, so only the low six bits of each byte value are used.

For example, character on Figure 5 is stored on the RAM address 0x00: “\texttt{ESC[14;31;21;31;23;16;31;14;0d}”. Since microcontroller read commands as ASCII codes, this command is sent as: “\texttt{0x1B 0x5B 0x33 0x70}”. Then, to display this character, send the numerical value of the addressed character. New character is performed by writing 0x00. The character will appear at the location of the LCD’s cursor.

![User defined character pattern](image)

Figure 5 User defined character pattern

This character is saved on the address 0x00 in the LCD’s RAM table. Therefore, RAM table must be selected as the main character. This is achieved by sending the following command: “\texttt{ESC[<pt>p}”; in ASCII type as: “\texttt{0x1B 0x5B 0x33 0x70}”. Then, to display this character, send the numerical value of the addressed character. New character is performed by writing 0x00. The character will appear at the location of the LCD’s cursor.
These characters can be also stored in EEPROM. Before storing any value to an address in EEPROM, writing to EEPROM must first be enabled. The command to enable writing to the EEPROM must be sent before each successive write to EEPROM. If the command to enable writing to the EEPROM is not sent before an EEPROM-related command, the command will be ignored.

In this design, 0x00 address is used as invalid character to show end of the data in ROM. User defined characters pattern and address are shown in Figure 6.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>27</td>
<td>31</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>26</td>
<td>28</td>
<td>24</td>
</tr>
<tr>
<td>31</td>
<td>14</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

Addressed 0x01

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>30</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>26</td>
<td>11</td>
<td>7</td>
<td>21</td>
</tr>
<tr>
<td>24</td>
<td>3</td>
<td>7</td>
<td>31</td>
</tr>
<tr>
<td>28</td>
<td>31</td>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>14</td>
<td>31</td>
<td>3</td>
<td>31</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
</tbody>
</table>

Addressed 0x02

Addressed 0x04

Addressed 0x05

Figure 6 Used characters in design

**TOP_MODULE**

Top Module consists of four modules. Two of them are FSM (Finite-State Machine) modules, one of them is a ROM memory module, and the other one is a clock divider.

Connections between modules are shown in Figure 7. An external oscillator on IC6 (UCLK) socket is used to generate 50MHz main clock. Clock divider module uses the main clock to generate 195.312 kHz.
Figure 7 Top Module

**FSM_SPI Module**

FSM_SPI module converts 8-bit parallel data to serial data. It also generates SCK for synchronization, and SS to select slave device. The flow chart of FSM_SPI is shown in the Figure 8.

SCLK and DONE signals are set ‘1’ and ‘0’ respectively. Then START input signal is checked. This signal is generated by FSM_CTRL module. It means that data has been read from the ROM, data is valid at FSM_SPI input and ready to write to LCD. If START is ‘0’, SS is kept at high till START signal becomes high. When START is ‘1’, counter is cleared and data sending is allowed by setting SS to ‘0’. Then DATA signal is copied to TEMP register, after that SCLK is inverted. In this way, a falling edge of SCLK is generated for shifting data. That also explains why the SCLK’s value kept high in the beginning. After that,
MSB of the TEMP is copied to MOSI line and TEMP register is shifted left by one bit. Then SCLK is inverted again, in this way, a rising edge of SCLK is generated and data sampled by the slave device. Counter is used to track number of sent bits. When LSB of TEMP is sampled, counter is reached its maximum value and DONE signal becomes high. It means that the 8 bit DATA is sent successfully. At “completed” state, DONE signal is kept high so FSM_CTRL module has enough time to bring START signal low. During that time no data transferring is performed. SCLK is used in FSM for generate and copied to SCK output.

Figure 8 Flow chart of FSM_SPI module
FSM_CTRL Module

FSM_CTRL module reads the data from ROM module. It also sends a START signal to FSM_SPI to start data sampling. The flow chart of FSM_CTRL is shown in Figure 9.

Initial value of ADDRESS and START signals are set to ‘0’. ADDRESS is an output signal which specifies the ROM’s address. After data on the ROM specified by address is copied to DATA register, DATA is checked whether it is valid or not at “checking” state. When DATA is 0x00, it means DATA isn’t valid. In this case, START signal is set to ‘0’.

If DATA signal has a valid value and DATA isn’t an “ESC” character, then START signal is set ‘1’ and sent to FSM_SPI module. Until DONE signal becomes high at FSM_SPI module, new data isn’t read from ROM. When DONE signal is ‘1’, START signal goes to low, and ADDRESS is increased in order to read the next data from ROM.

Animation of characters requires slowing down printing of characters. However, this is time consuming if commands are sent. Therefore, two procedures are required one for sending characters and the other one for sending commands. If delay is not required, state follows “command”, “parameters” and “completed”. When delay is required state goes to delay state “delay” after going to “send”. During a delay state, START signal is kept low. When delay counter “q” is reached its maximum value, state is returned to “checking” and new data is read.
ROM_MEMORY Module

ROM is a kind of another interface between user and design. User applies data to ROM; design reads the user’s data from ROM. In this design, ROM_Memory module is written in Verilog language. Creating a ROM block in Verilog has an advantage since Verilog can read data directly from a .txt file.

CLOCK_DIVIDER Module

SPI communication needs SCK clock either read or write data synchronously. SCK clock is generated by FSM_SPI. Since frequency of SCK is selected as 97.66 kHz, FSM_SPI must be clocked at twice of this frequency. Clock divider block has an eight bit counter [7:0]
which is clocked at 50 MHz frequency. MSB of that counter has a 195.312 kHz frequency.
FSM_CTRL and FSM_SPI modules are running with this 195.312 kHz clock.

**SIMULATIONS**

ISIM provided by Xilinx ISE Design Suite is used to simulate and validate the design. The verification method consists of simulation of individual blocks and testing various commands by top level simulations.

FSM_SPI module, which is responsible for data transfer, is the first designed part. Outputs of this module are SS, MOSI and SCK for generating SPI signals. End of operation is indicated by DONE signal. Inputs of this module are an 8-bit DATA signal for reading data and a begin operation signal indicated by START signal.

When START signal becomes high, DATA is copied to the TEMP register. Then the value at the TEMP registers is put at every falling edge of SCK on the MOSI output. A counter tracks number of sent bits. Once DATA transferring is completed successfully, DONE signal becomes high. SS signal depends on the START only at the beginning of whole operation. Its transition from low to high occurs when the operation is finished. Data transferring is verified with simulations of FSM_SPI module and the results are shown in Figure 10 and Figure 11.

FSM_CTRL module is responsible for reading data from ROM and sending a start and DATA signals to FSM_SPI module. Outputs of this module are ADDRESS and START signal. ADDRESS specifies data at ROM. Start of operation indicates with START. Inputs of this module are DATA and DONE signal. DONE is used to increase ADDRESS linearly for every successful read from ROM. Simulation results of generating a START signal and delays are shown in Figure 12. Behavior for invalid character is verified and results are shown in Figure 13.

Top module is also simulated to verify connections between sub-modules. Sending a character and sending character successively is shown in Figure 14, Figure 15 respectively. Successive sending of an escape sequence without delay is shown in Figure 16. Delay processing, when a character is detected, is shown in Figure 17 and Figure 18.
Figure 10 Simulation of FSM_SPI module (states of fsm)

Figure 11 Simulation of FSM_SPI module (transfer complete)
Figure 12 Simulation of FSM_CTRL module character sending followed by escape sequences

Figure 13 Simulation of FSM_CTRL module invalid character
Figure 14 “ESC” character sending in address “0”

Figure 15 Successive sending of “ESC” and “[”
Figure 16 Escape sequence sending as a string

Figure 17 Delay is starting to work after a “SPACE” character written
Timing Closure

Main clock (mclk) is generated by an external oscillator; which is connected to IC6 (UCLK) socket. Clock specifications are defined in LCD.ucf file. 50 MHz clock which has 20ns period, has 50% duty cycle. Jitter is the time variation of a periodic signal as a clock signal. Although the external oscillator has maximum 1ps phase jitter, for timing closure 100 ps is selected in order to guarantee proper operation even worst conditions.

SCK is selected as 97.66 kHz, FSM_SPI and FSM_CTRL are clocked at twice of SCK’s frequency. Since data on MOSI is changed on falling edge of SCK and sampled at the rising edge of SCK, there is enough time for setup and hold of MOSI with respect to SCK. Therefore, timing at the SPI output is not required to be checked.

Figure 18 “O” character written after delay, and delaying again
REFERENCES

http://www.digilentinc.com/Data/Products/BASYS2/Basys2_rm.pdf


http://www.digilentinc.com/Data/Products/PMOD-CLS/PmodCLS_sch.pdf

http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,66,828&Prod=ADEPT2
  - DSPI Programmer’s reference manual.pdf (digilent.adpt.sdk_v2.0.1/doc)

http://www.digilentinc.com/classroom/CerebotLabSet/LabProject7/LabProject7.pdf


http://www.asciiitable.com/

Okan University / Faculty of Engineering and Architecture / Department of Electrical and Electronics Engineering / Digital Design Lecture Notes by Asst. Prof. Burak Kelleci (burak.kelleci@okan.edu.tr)
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
entity fsm_spi is
Port ( clk : in  STD_LOGIC;
  clr : in std_logic;
  data : in  STD_LOGIC_VECTOR(7 downto 0);
  start : in  STD_LOGIC;
  done : out  STD_LOGIC;
  ss : out  STD_LOGIC;
  mosi : out  STD_LOGIC;
  sck : out  STD_LOGIC);
end fsm_spi;

architecture fsm_spi of fsm_spi is

signal counter: STD_LOGIC_VECTOR (3 downto 0);
signal sclk: std_logic;
signal temp: std_logic_vector(7 downto 0);

type state_type is(initial,copy,clk_gen,shifting,completed);
signal state: state_type;

attribute INIT: STRING;
attribute INIT OF state: SIGNAL IS "initial";

begin
process (clk, clr)
begin
  if clr = '1' then
    done <= '0';
    ss <= '1';
    mosi <= '0';
    sclk <= '1';
    counter <= "0000";
    state <= initial;
  elsif clk'event and clk = '1' then
    case state is
    when initial =>
      if start = '0' then
        state <= completed;
      else
        counter <= "0000";
        state <= copy;
        ss <= '0';
      end if;
      sclk <= '1';
      done <= '0';
    when copy =>
      temp <= data;
      state <= clk_gen;
    when clk_gen =>
      sclk <= not sclk;
    when

end process;

end process;
state <= shifting;
when shifting =>
  if counter = "1000" then
    done <= '1';
    state <= completed;
  else
    counter <= counter + '1';
    state <= clk_gen;
  end if;
  mosi <= temp(7);
  temp <= (temp(6 downto 0) & '0');
  sclk <= not sclk;
when completed =>
  ss <= '1';
  state <= initial;
when others =>
  state <= initial;
end case;

sck <= sclk;
end if;
end process;
end fsm_spi;

FSM_CTRL.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
entity fsm_ctrl is
  Port ( clr : in  STD_LOGIC;
         clk : in  STD_LOGIC;
         address : buffer  STD_LOGIC_vector(10 downto 0);
         data : in  STD_LOGIC_vector(7 downto 0);
         start : out  STD_LOGIC;
         done : in  STD_LOGIC);
end fsm_ctrl;

architecture fsm_ctrl of fsm_ctrl is
  signal q : std_logic_vector(14 downto 0);
  type state_type is
    (initial,checking,send,delay,command,parameters,completed);
  signal state: state_type;
begin
  process (clk, clr)
  begin
    if clr = '1' then
      start <= '0';
      address <= (others => '0');
      state <= initial;
      q <= (others => '0');
    elsif clk'event and clk = '1' then
      case state is
        when initial =>
          state <= checking;
          q <= (others => '0');
        when checking =>
          if counter = "1000" then
            done <= '1';
            state <= completed;
          else
            counter <= counter + '1';
            state <= clk_gen;
          end if;
          mosi <= temp(7);
          temp <= (temp(6 downto 0) & '0');
          sclk <= not sclk;
        when completed =>
          ss <= '1';
          state <= initial;
        when others =>
          state <= initial;
      end case;
      sck <= sclk;
    end if;
  end process;
end fsm_ctrl;
case state is
when initial =>
    address <= (others => '0');
    start <= '0';
    state <= checking;
when checking =>
    if data = X"00" then --data is checking.
        start <= '0';
        address <= (others => '0');
        --if address's value won’t be ‘0’, when data’s
        --value "00" system will wait for reset signal
        --to restart.
        state <= checking;
    elsif data = X"1B" then --escape sequence checking.
        state <= command;
    else
        start <= '1';
        state <= send;
    end if;
when send =>
    if done = '1' then
        start <= '0';
        address <= address + '1';
        q <= (others => '0');
        --if delay is not necessary, state should return
        --to state “checking” at here.
        state <= delay;
    else
        state <= send;
    end if;
when delay => --delay state for characters send.
    if q = "111111111111111" then
        state <= checking;
    else
        q <= q + '1';
        state <= delay;
    end if;
when command =>
    start <= '1';
    case data is
    --commands list for understand the escape sequence ending.
    when X"48" => state <= completed; --H command
    when X"73" => state <= completed; --s command
    when X"75" => state <= completed; --u command
    when X"6A" => state <= completed; --j command
    when X"4B" => state <= completed; --K command
    when X"4E" => state <= completed; --N command
    when X"40" => state <= completed; --@ command
    when X"41" => state <= completed; --A command
    when X"2A" => state <= completed; --* command
    when X"65" => state <= completed; --e command
    when X"68" => state <= completed; --h command
    when X"63" => state <= completed; --c command
    when X"61" => state <= completed; --a command
    when X"62" => state <= completed; --b command
    when X"70" => state <= completed; --p command
    when X"74" => state <= completed; --t command
    when X"6C" => state <= completed; --l command
    when X"64" => state <= completed; --d command
    when X"6D" => state <= completed; --m command
    when X"77" => state <= completed; --w command
when X"6E" => state <= completed; -- n command
when X"6F" => state <= completed; -- o command
when others => state <= parameters;
-- while command comes, continue to read and write
-- escape sequences
end case;
when parameters =>
  if done = '1' then
    start <= '0';
    address <= address + '1';
    state <= command;
  else
    state <= parameters;
  end if;
when completed =>
  if done = '1' then
    start <= '0';
    address <= address + '1';
    state <= checking;
  else
    state <= completed;
  end if;
when others =>
  state <= initial;
end case;
end if;
end process;
end fsm_ctrl;

ROM_Memory.v

`timescale 1ns / 1ps
module ROM_Memory(data,address);
  input [10:0] address;
  output [7:0] data;
  reg [7:0] ROM [0:1034];
  initial begin
    $readmemh("ROM.txt",ROM);
  end
  assign data = ROM[address];
endmodule

CLOCK_DIV.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
entity clock_div is
  Port ( clr : in  STD_LOGIC;
         mclk : in  STD_LOGIC;
         clk_out : out  STD_LOGIC);
end clock_div;

architecture clock_div of clock_div is
  signal q: std_logic_vector(7 downto 0);
begin
process(mclk, clr)
begin
if clr = '1' then
  q <= (others => '0');
  clk_out <= '0';
elsif mclk'event and mclk = '1' then
  if q = "1111111" then
    q <= (others => '0');
  else
    q <= q + '1';
  end if;
  clk_out <= q(7);
end if;
end process;
end clock_div;

TOP_MODULE.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity top_module is
  Port ( clr : in  STD_LOGIC;
         clk : in  STD_LOGIC;
         ss : out  STD_LOGIC;
         mosi : out  STD_LOGIC;
         sck : out  STD_LOGIC);
end top_module;

architecture top_module of top_module is

component clock_div is
  Port ( clr : in  STD_LOGIC;
         mclk : in  STD_LOGIC;
         clk_out : out  STD_LOGIC);
end component;

component ROM_Memory is
  port(address: in std_logic_vector(10 downto 0);
       data: out std_logic_vector(7 downto 0));
end component;

component fsm_ctrl is
  Port ( clr : in  STD_LOGIC;
        clk : in  STD_LOGIC;
        address : buffer  STD_LOGIC_vector(10 downto 0);
        data : in  STD_LOGIC_vector(7 downto 0);
        start : out  STD_LOGIC;
        done : in  STD_LOGIC);
end component;

component fsm_spi is
  Port ( clk : in  STD_LOGIC;
        clr : in std_logic;
        data : in  STD_LOGIC_VECTOR(7 downto 0);
        start : in  STD_LOGIC;
        done : out  STD_LOGIC;

ss : out  STD_LOGIC;
mosi : out  STD_LOGIC;
sck : out  STD_LOGIC);
end component;

signal send_data : std_logic_vector(7 downto 0);
signal send_address : std_logic_vector(10 downto 0);
signal send_start, send_done, send_clk_out : std_logic;

begin

U_clock_div: clock_div port map (clr => clr, mclk => clk, clk_out =>
send_clk_out);
U_fsm_ctrl: fsm_ctrl port map (clr => clr, clk => send_clk_out, address =>
send_address, start => send_start, done => send_done, data => send_data);
U_fsm_spi: fsm_spi port map (clr => clr, clk => send_clk_out, data =>
send_data, start => send_start, done => send_done, ss => ss, mosi => mosi,
sck => sck);
U_rom: ROM_Memory port map (address => send_address, data => send_data);
end top_module;

LCD.ucf

# clock pin for Basys2 Board
NET "clk" LOC = "M6"; # Bank = 2, Signal name = UCLK
NET "clk" TNM_NET = mclk;
TIMESPEC TS_mclk = PERIOD "mclk" 20 ns HIGH 50%;
SYSTEM_JITTER = 100 ps;
# Pin assignment for SWs
NET "clr" LOC = "G12"; # Bank = 0, Signal name = BTN0
# Loop Back only tested signals
NET "ss" LOC = "B2" ; #| DRIVE = 2 | PULLUP ; # Signal name = JA1
NET "mosi" LOC = "A3" ; #| DRIVE = 2 | PULLUP ; # Signal name = JA2
#NET "miso" LOC = "J3" ; #| DRIVE = 2 | PULLUP ; # Signal name = JA3
NET "sck" LOC = "B5" ; #| DRIVE = 2 | PULLUP ; # Signal name = JA4

ROM.txt

1B, 5B, 30, 65 ( enable/disable display; display off, backlight off )
1B, 5B, 30, 6A ( clear display and home cursor )
1B, 5B, 30, 63 ( set cursor mode; cursor off )
1B, 5B, 30, 68 ( set display mode; wrap line at 16 characters )
20 ( space )
1B, 5B, 30, 3B, 30, 48 ( set cursor position to 0 row, 0 column )
20 ( space )
4F ( O )
4B ( K )
41 ( A )
4E ( N )
20 ( space )
55 ( U )
4E ( N )
49 ( I )
56 ( V )
45 ( E )
52 ( R )
53 ( S )
49 ( I )
54 ( T )
59 ( Y )
1B, 5B, 30, 3B, 30, 48 ( set cursor position to 0 row, 0 column )
1B, 5B, 34, 3B, 31, 34, 3B, 32, 37, 3B, 33, 31, 3B, 32, 34, 3B, 33, 31, 3B, 31, 34, 3B, 34, 3B, 31, 64 ( define first character at address 0x01 )
1B, 5B, 36, 3B, 31, 35, 3B, 32, 36, 3B, 32, 38, 3B, 32, 34, 3B, 32, 38, 3B, 31, 35, 3B, 36, 3B, 32, 64 ( define second character at address 0x02 )
1B, 5B, 31, 32, 3B, 33, 30, 3B, 31, 31, 3B, 37, 3B, 33, 3B, 37, 3B, 33, 30, 3B, 31, 32, 3B, 34, 64 ( define third character at address 0x04 )
1B, 5B, 33, 70 ( program character table 3 into LCD )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 30, 48 ( set cursor position to 0 row, 0 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 30, 48 ( set cursor position to 0 row, 0 column )
20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 31, 48 ( set cursor position to 0 row, 1 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 31, 48 ( set cursor position to 0 row, 1 column )
20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 32, 48 ( set cursor position to 0 row, 2 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 32, 48 ( set cursor position to 0 row, 2 column )
20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 33, 48 ( set cursor position to 0 row, 3 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 33, 48 ( set cursor position to 0 row, 3 column )
20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 34, 48 ( set cursor position to 0 row, 4 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 34, 48 ( set cursor position to 0 row, 4 column )
20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 35, 48 ( set cursor position to 0 row, 5 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 35, 48 ( set cursor position to 0 row, 5 column )
20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 36, 48 ( set cursor position to 0 row, 6 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 36, 48 ( set cursor position to 0 row, 6 column )
20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 37, 48 ( set cursor position to 0 row, 7 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 37, 48 ( set cursor position to 0 row, 7 column )
20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 38, 48 ( set cursor position to 0 row, 8 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 38, 48 ( set cursor position to 0 row, 8 column )
20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 39, 48 ( set cursor position to 0 row, 9 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 39, 48 ( set cursor position to 0 row, 9 column )

20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 31, 30, 48 ( set cursor position to 0 row, 10 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 31, 30, 48 ( set cursor position to 0 row, 10 column )

01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 31, 30, 48 ( set cursor position to 0 row, 10 column )

20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 31, 31, 48 ( set cursor position to 0 row, 11 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 31, 31, 48 ( set cursor position to 0 row, 11 column )

20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 31, 32, 48 ( set cursor position to 0 row, 12 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 31, 32, 48 ( set cursor position to 0 row, 12 column )

01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 31, 32, 48 ( set cursor position to 0 row, 12 column )

20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 31, 33, 48 ( set cursor position to 0 row, 13 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 31, 33, 48 ( set cursor position to 0 row, 13 column )

20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 31, 34, 48 ( set cursor position to 0 row, 14 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 31, 34, 48 ( set cursor position to 0 row, 14 column )

01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 31, 34, 48 ( set cursor position to 0 row, 14 column )

20 ( space )
02 ( write user defined character at address 0x02 )
1B, 5B, 30, 3B, 31, 35, 48 ( set cursor position to 0 row, 15 column )
01 ( write user defined character at address 0x01 )
1B, 5B, 30, 3B, 31, 35, 48 ( set cursor position to 0 row, 15 column )

20 ( space )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 31, 36, 48 ( set cursor position to 0 row, 16 column )

20 ( space )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 31, 37, 48 ( set cursor position to 0 row, 17 column )

20 ( space )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 31, 38, 48 ( set cursor position to 0 row, 18 column )
04 ( write user defined character at address 0x04 )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 31, 39, 48 ( set cursor position to 0 row, 19 column )

20 ( space )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 32, 30, 48 ( set cursor position to 0 row, 20 column )

20 ( space )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 32, 31, 48 ( set cursor position to 0 row, 21 column )
05 ( write user defined character at address 0x05 )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 32, 32, 48 ( set cursor position to 0 row, 22 column )

20 ( space )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 32, 33, 48 ( set cursor position to 0 row, 23 column )

20 ( space )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 32, 34, 48 ( set cursor position to 0 row, 24 column )
4F ( O )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 32, 35, 48 ( set cursor position to 0 row, 25 column )
4B ( K )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 32, 36, 48 ( set cursor position to 0 row, 26 column )
41 ( A )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 32, 37, 48 ( set cursor position to 0 row, 27 column )
4E ( N )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 32, 38, 48 ( set cursor position to 0 row, 28 column )
20 ( space )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 32, 39, 48 ( set cursor position to 0 row, 29 column )
55 ( U )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 33, 30, 48 ( set cursor position to 0 row, 30 column )
4E ( N )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 33, 31, 48 ( set cursor position to 0 row, 31 column )
49 ( I )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 33, 32, 48 ( set cursor position to 0 row, 32 column )
56 ( V )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 33, 33, 48 ( set cursor position to 0 row, 33 column )
45 ( E )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 33, 34, 48 ( set cursor position to 0 row, 34 column )
52 ( R )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 33, 35, 48 ( set cursor position to 0 row, 35 column )
53 ( S )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 33, 36, 48 ( set cursor position to 0 row, 36 column )
49 ( I )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 33, 37, 48 ( set cursor position to 0 row, 37 column )
54 ( T )
1B, 5B, 31, 40
1B, 5B, 30, 3B, 33, 38, 48 ( set cursor position to 0 row, 38 column )
59 ( Y )
20 ( space )
1B, 5B, 30, 3B, 33, 38, 48 ( set cursor position to 0 row, 38 column )
20 ( space )
1B, 5B, 31, 3B, 33, 38, 48 ( set cursor position to 1 row, 38 column )
59 ( Y )
1B, 5B, 30, 3B, 33, 37, 48 ( set cursor position to 0 row, 37 column )
20 ( space )
1B, 5B, 31, 3B, 33, 37, 48 ( set cursor position to 1 row, 37 column )
54 ( T )
1B, 5B, 30, 3B, 33, 36, 48 ( set cursor position to 0 row, 36 column )
20 ( space )
1B, 5B, 31, 3B, 33, 36, 48 ( set cursor position to 1 row, 36 column )
49 ( I )
1B, 5B, 30, 3B, 33, 35, 48 ( set cursor position to 0 row, 35 column )
20 ( space )
1B, 5B, 31, 3B, 33, 35, 48 ( set cursor position to 1 row, 35 column )
53 ( S )
1B, 5B, 30, 3B, 33, 34, 48 ( set cursor position to 0 row, 34 column )
1B, 5B, 31, 3B, 32, 39, 48 (set cursor position to 1 row, 29 column)
20 (space)
1B, 5B, 31, 3B, 32, 37, 48 (set cursor position to 1 row, 27 column)
20 (space)
1B, 5B, 31, 3B, 32, 36, 48 (set cursor position to 1 row, 26 column)
20 (space)
1B, 5B, 31, 3B, 32, 35, 48 (set cursor position to 1 row, 25 column)
20 (space)
1B, 5B, 31, 3B, 32, 34, 48 (set cursor position to 1 row, 24 column)
20 (space)
00 (null)