Contents

Chapter 1 HAHA Board.................................................................................................................. 1
  1.1 Layout and components ........................................................................................................ 1
  1.2 Block Diagram of the HAHA Board ...................................................................................... 3
Chapter 2 Using the HAHA Board ............................................................................................... 6
  2.1 Power up the HAHA Board ................................................................................................. 6
  2.2 Configuring the Altera MAX 10 FPGA ................................................................................ 6
  2.3 Configuring the Micro controller through USB port ............................................................... 8
  2.4 Configuring the voltage source ........................................................................................... 9
  2.5 Chip interconnections ......................................................................................................... 9
  2.6 Using the switches and LEDs ............................................................................................ 9
  2.7 Using the 7-segment display .............................................................................................. 11
  2.8 Using the expansion headers ............................................................................................. 11
  2.9 Clock circuitry ..................................................................................................................... 13
  2.10 Using SPI devices ............................................................................................................ 14
Chapter 3 Examples for Programming ....................................................................................... 16
  3.1 Programming the FPGA ....................................................................................................... 16
    3.1.1 Counter implemented in the FPGA ................................................................................. 16
    3.1.2 Arithmetic and Logic Unit ............................................................................................. 18
    3.1.3 Finite State Machine ..................................................................................................... 19
    3.1.4 Read-write operation of an SRAM ................................................................................. 23
  3.2 Programming the Micro controller ....................................................................................... 24
    3.2.1 Counter programmed in the Micro controller ................................................................. 24
    3.2.2 Light up a LED through an I/O port .............................................................................. 26
    3.2.3 Make the LED to blink .................................................................................................. 27
    3.2.4 Read a value from a SRAM address and send it to I/O ports .......................................... 28
    3.2.5 Program controlled by an input port .............................................................................. 28
Chapter 1 HAHA Board

This chapter presents the features and design characteristics of the HAHA Board (Hardware Hacking Board).

1.1 Layout and components

A photograph of the HAHA board is shown in Figure 1. It depicts the layout of the board and indicates the location of the connectors and key components.

![Figure 1 The HAHA board.](image)

The HAHA board has many features that allow the user to implement a wide range of designed circuits, and to conduct hardware hacking experiments.

The following hardware is provided on the HAHA board:

- Altera MAX 10 FPGA device
- Atmel AVR 8-bit Micro controller
- JTAG header for programming both the FPGA and Micro controller
- SPI (Serial Peripheral Interface) header for programming the Micro controller
- USB B port for programming the Micro controller
- High performance 3-axis accelerometer
- 1 Mbit serial EEPROM memory
- Bluetooth specification v4.0 compliant Bluetooth network processor
- Adjustable regulator providing power source from 2.470V to 4.860V
- 2 toggle switches selecting power source from fixed 3.3V and adjustable source
- 1 Ohm current sensing resistor and instrumentation amplifier
- 3 pushbutton switches
- 8 slide switches
- 8 green user LEDs
- 50-MHz and 8-MHz oscillator for clock sources
- Two IO pin expansion headers
- 7 segment display
- Photodiode
1.2 Block Diagram of the HAHA Board

Figure 2 gives the diagram of the HAHA board. Devices that can be configured through SPI are connected with the micro controller. The micro controller can program them and communicate with them as the SPI master. Other connections are made through the Altera MAX 10 FPGA device. There are 9 direct connections between the FPGA and the micro controller. All of them can be used as channels for data and one of them can also be the clock source for the FPGA. In this way, both the FPGA and the micro controller have access to all the peripherals. They can share functions and work as a whole.

Following is the detailed information about the blocks in Figure 2.

- Altera 10M50SCE144C8G FPGA
  - 50k logic elements
  - 1638k M9k Memory
  - 5888k user Flash Memory
  - 101 user I/O pins
  - 4 PLLs
  - Boundary-scan capabilities according to the JTAG standard
- Atmel ATmega16U4-au Micro controller
  - 16KB of in-system self-programmable flash
  - 1.25KB internal SRAM
  - Boundary-scan capabilities according to the JTAG standard
  - USB 2.0 full-speed/low-speed device module
  - SPI serial programmable

- EEPROM
  - 256-byte page
  - Built-in write protection
  - 6 ms max. write cycle time

- JTAG header
  - On board header for programming both the FPGA and Micro controller

- SPI header
  - On board header for programming the Micro controller
  - Can be used as expansion slots for add-on SPI modules

- USB B port
  - On board USB B port for programming the Micro controller
  - Provide power source for the board

- High performance 3-axis accelerometer
  - SPI digital output interface
  - 8-bit data output
  - Motion detection
  - Embedded temperature sensor
  - Embedded FIFO

- Bluetooth processor
  - Bluetooth v4.0 compliant
  - SPI based application controller interface
  - AES security co-processor

- Adjustable regulator
  - Providing power source from 2.6V to 5V
  - 64 steps resolution
  - 500 mA current capability
  - Two push buttons controlling: one for making voltage higher and the other lower

- 2 slide switches selecting power source
Select from fixed 3.3V and adjustable source
When the switch is “OFF”, the power source will be fixed 3.3V
When the switch is “ON”, the power source will be the adjustable voltage source

Current sensing resistor
- Resistance 1 Ohm
- Tolerance 0.5%
- Max. current 20A
- Instrumentation amplifier included to the power supply lines, across the sense resistors

Pushbutton switches
- 3 pushbutton switches
- Normally high; generates one active-low pulse when the switch is pressed

Slide switches
- 8 Slide switches
- A switch causes logic 0 when in the ON (Left) position and logic 1 when in the OFF (Right) position

User LEDs
- 8 Green LEDs connected to the FPGA
- 1 Red LED connected to the Micro controller
- The Red LED can be controlled by an 8-bit timer/counter with PWM (Pulse Width Modulator)

Clock inputs
- 8-MHz oscillator for the Micro controller
- 50-MHz oscillator for the FPGA
- The Micro controller can share its clock through the chip interconnection
- Both chips have their internal clocks

Two IO pin expansion headers
- Connected to the IO pins of the FPGA
- One of them has 40 pins
- The other has 20 pins

7 segment display
- One 7 segment display connected to the FPGA

Photodiode
- One photodiode connected to the ADC (Analog-to-digital converter) pin of the Micro controller
- Peak Wavelength 890 nm
Chapter 2 Using the HAHA Board

This chapter gives instructions for using the HAHA board and describes each of its devices.

2.1 Power up the HAHA Board

The HAHA Board comes with a USB A to B cable, which will connect the HAHA board to computer or USB power port. To power up the board, perform the following steps.

1. Make sure the power ON/OFF switch on the HAHA board is OFF, and 4 screw stands on the corners are mounted. Place the board at a clean place; make sure there are no conductive connections underneath the board that may cause a short.
2. Select the power source for the FPGA and the Micro controller. Generally, you need to turn OFF both the selecting switches to make the power source to be fixed 3.3V. Turn OFF the reset switch for the Micro controller.
3. Plug in the USB cable Type-B port to the board and Type-A port to a computer.
4. Turn on the power ON/OFF switch. A red LED will be on, indicating the power is on.

2.2 Configuring the Altera MAX 10 FPGA

The HAHA Board comes with a USB-Blaster Download Cable which can be used to program the Altera MAX 10 FPGA. As shown in Figure 3, one side of the USB-Blaster will be connected to a computer USB Port, the other side will be connected to the JTAG 10-pin header on the board.

Make sure that the 10-pin female connector (Figure 4) and the JTAG header on the HAHA Board are connected correctly. Pin numbers are marked on the HAHA Board. Connect it accordingly. Note the red wire in the band is for pin 1.
After you connect the USB-Blaster Download Cable properly between the HAHA board and a computer, you should be able to find the 2 chips in the JTAG chain on the board: the FPGA and the Micro controller. In the Program window of Quartus (Figure 5), you can “Auto Detect” and see two devices are detected: the FPGA and the Micro controller. Replace the FPGA with the SOF-file that you made, then you can start to program the FPGA. Note that FPGA will lose all of the configuration after power off, therefore you have to program it every time you restart the board.

![Figure 4 10-pin Female Connector](image)

![Figure 5 Program window of Quartus, auto detected the FPGA and the Micro controller](image)
2.3 Configuring the Micro controller through USB port

The Atmel AVR Micro controller ATmega16U4-au can be configured in many ways including JTAG, SPI, USB, etc. Among them, the JTAG and SPI method will require an Atmel Debugger, which will not come along with the HAHA Board. You can refer to Atmel website to find more information about programming the AVR Micro controller using different kinds of debuggers.

You can directly program the 16KB in-system self-programmable flash through the USB port. After you power up the HAHA board, you detect the Micro controller in the Atmel Studio. Open the Device Programming window as shown in Figure 6, you can detect the Micro controller that is connecting to the computer. The device signature can be read, and the whole chip can be erased here.

Program the programmable flash by selecting the HEX-file you write and pressing the program button.

![Figure 6 Atmel Studio Device Programming window, the Micro controller is detected](image)

Note that the content of the flash will not be gone after the power is OFF. Therefore, you can use same function many times until you reset the chip by turning on the reset switch. Also, the function you just programmed will not work once you finished programming. You have to power the board OFF and then ON to make the functions you programmed work.

Due to this difference, when you need to program both the Micro controller and the FPGA, always program the Micro controller first, power OFF and ON the board, and then program the FPGA.
2.4 Configuring the voltage source

Both the FPGA and the Micro controller will only need a 3.3V power supply. The HAHA board provides two voltage sources: fixed 3.3V and adjustable voltage. Each of the FPGA and the Micro controller has one slide switch to select the voltage source. You can either use the fixed 3.3V or the adjustable voltage.

For most of the time, you will only need the fixed 3.3V voltage source to get the best chip performance. Therefore, please make sure you turn OFF both the selecting switches before you power up the HAHA Board. When you are doing a Hardware Hacking experiment, such as fault injection experiment, you may need to use the adjustable voltage source. At that time, turn on the voltage source selecting switch before you power up the board.

The adjustable voltage has a range from 2.470V to 4.860V. You can control the voltage by pressing two buttons: SWup and SWdown. There are in total 64 steps. The regulator can memorize the voltage value when the power is OFF. It is recommended that every time you finished using the adjustable voltage source, make the voltage to be around 3.3V (neither too high or too low) before you power off the board.

2.5 Chip interconnections

Between the two major chips – the FPGA and the Micro controller – on the board, there are in total 9 interconnections.

Their pin numbers and functions are listed in Table 1.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Micro controller Pin No.</th>
<th>FPGA Pin No.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM0</td>
<td>PD0</td>
<td>14</td>
<td>Data</td>
</tr>
<tr>
<td>CM1</td>
<td>PD1</td>
<td>13</td>
<td>Data</td>
</tr>
<tr>
<td>CM2</td>
<td>PD2</td>
<td>12</td>
<td>Data</td>
</tr>
<tr>
<td>CM3</td>
<td>PD3</td>
<td>11</td>
<td>Data</td>
</tr>
<tr>
<td>CM4</td>
<td>PD4</td>
<td>8</td>
<td>Data</td>
</tr>
<tr>
<td>CM5</td>
<td>PD5</td>
<td>10</td>
<td>Data</td>
</tr>
<tr>
<td>CM6</td>
<td>PD6</td>
<td>7</td>
<td>Data</td>
</tr>
<tr>
<td>CM7</td>
<td>PD7</td>
<td>6</td>
<td>Data</td>
</tr>
<tr>
<td>CLK_inter</td>
<td>PC7</td>
<td>28</td>
<td>Data or clock</td>
</tr>
</tbody>
</table>

The two chips can communicate through the interconnections so that both chips can have access to all the devices on the HAHA Board. For example, if the Micro controller wants to control the 7-segment display, even though the 7-segment display is not directly connected to the Micro controller, the FPGA can be configured as an interface connecting the Micro controller and the 7-segment display so that Micro controller gets access to the 7-segment display.

2.6 Using the switches and LEDs

The HAHA Board provides 3 pushbutton switches, 8 slide switches and 8 LEDs, all connecting to the FPGA. There is only one LED connected to the Micro controller.
There are 3 push buttons connected to the FPGA. Each switch provides a high logic level (3.3 volts) when it is not pressed, and provides a low logic level (0 volts) when depressed. Table 2 lists the corresponding pin numbers on the FPGA.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>FPGA Pin No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW1</td>
<td>85</td>
<td>Push Button [1]</td>
</tr>
<tr>
<td>SW2</td>
<td>86</td>
<td>Push Button [2]</td>
</tr>
<tr>
<td>SW3</td>
<td>87</td>
<td>Push Button [3]</td>
</tr>
</tbody>
</table>

There are also 8 slide switches (sliders) on the HAHA Board. These switches are not debounced and are intended for use as level-sensitive data inputs to a circuit. Each switch is connected directly to a pin on the FPGA. When a switch is in the ON (Left) position, it provides a low logic level (0 volts) to the FPGA, and when the switch is in the OFF (right) position, it provides a high logic level (3.3 volts). Table 3 lists the pin numbers connected to the switches.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>FPGA Pin No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>30</td>
<td>Slide Switch [1]</td>
</tr>
<tr>
<td>S2</td>
<td>29</td>
<td>Slide Switch [2]</td>
</tr>
<tr>
<td>S3</td>
<td>27</td>
<td>Slide Switch [3]</td>
</tr>
<tr>
<td>S4</td>
<td>26</td>
<td>Slide Switch [4]</td>
</tr>
<tr>
<td>S5</td>
<td>25</td>
<td>Slide Switch [5]</td>
</tr>
<tr>
<td>S6</td>
<td>23</td>
<td>Slide Switch [6]</td>
</tr>
<tr>
<td>S7</td>
<td>22</td>
<td>Slide Switch [7]</td>
</tr>
<tr>
<td>S8</td>
<td>21</td>
<td>Slide Switch [8]</td>
</tr>
</tbody>
</table>

There are 8 user-controllable LEDs on the HAHA Board. Each LED is driven directly by a pin on the FPGA; driving its associated pin to a high logic level turns the LED on, and driving the pin low turns it off. Table 4 lists the pin numbers connected to the LEDs.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>FPGA Pin No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>141</td>
<td>LED Green [1]</td>
</tr>
<tr>
<td>D2</td>
<td>140</td>
<td>LED Green [2]</td>
</tr>
<tr>
<td>D3</td>
<td>135</td>
<td>LED Green [3]</td>
</tr>
<tr>
<td>D4</td>
<td>133</td>
<td>LED Green [4]</td>
</tr>
<tr>
<td>D5</td>
<td>132</td>
<td>LED Green [5]</td>
</tr>
<tr>
<td>D6</td>
<td>131</td>
<td>LED Green [6]</td>
</tr>
<tr>
<td>D7</td>
<td>129</td>
<td>LED Green [7]</td>
</tr>
<tr>
<td>D8</td>
<td>127</td>
<td>LED Green [8]</td>
</tr>
</tbody>
</table>
2.7 Using the 7-segment display

The HAHA Board has one 7-segment display. As indicated in Figure 7, the seven segments are connected to pins on the FPGA. Applying a low logic level to a segment causes it to light up, and applying a high level turns it off.

Each segment in the display is identified by an index from a to g, with the position given in Figure 7. In addition, the decimal point is identified as DP. Table 5 shows the connections between the FPGA pins to the 7-segment display.

![Figure 7 Connections between the 7-segment display and the FPGA](image)

**Table 5 Pin assignments for the 7-segment displays**

<table>
<thead>
<tr>
<th>Display Index</th>
<th>Signal Name</th>
<th>FPGA Pin No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>D11</td>
<td>126</td>
</tr>
<tr>
<td>b</td>
<td>D12</td>
<td>123</td>
</tr>
<tr>
<td>c</td>
<td>D13</td>
<td>120</td>
</tr>
<tr>
<td>d</td>
<td>D14</td>
<td>119</td>
</tr>
<tr>
<td>e</td>
<td>D15</td>
<td>118</td>
</tr>
<tr>
<td>f</td>
<td>D16</td>
<td>117</td>
</tr>
<tr>
<td>g</td>
<td>D17</td>
<td>116</td>
</tr>
<tr>
<td>DP</td>
<td>D18</td>
<td>111</td>
</tr>
</tbody>
</table>

2.8 Using the expansion headers

The HAHA Board provides two expansion headers. One of them has 40 pins, and the other has 20 pins. In total 53 IO pins of the FPGA are connected to the header pins, and the lists are shown in Table 6 and Table 7.

**Table 6 Expansion header 1 pin number list**

<table>
<thead>
<tr>
<th>Header Pin No.</th>
<th>FPGA Pin No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>GND</td>
</tr>
<tr>
<td>H2</td>
<td>GND</td>
</tr>
<tr>
<td>H3</td>
<td>32</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
</tr>
<tr>
<td>H4</td>
<td>33</td>
</tr>
<tr>
<td>H5</td>
<td>39</td>
</tr>
<tr>
<td>H6</td>
<td>38</td>
</tr>
<tr>
<td>H7</td>
<td>41</td>
</tr>
<tr>
<td>H8</td>
<td>40</td>
</tr>
<tr>
<td>H9</td>
<td>43</td>
</tr>
<tr>
<td>H10</td>
<td>42</td>
</tr>
<tr>
<td>H11</td>
<td>46</td>
</tr>
<tr>
<td>H12</td>
<td>44</td>
</tr>
<tr>
<td>H13</td>
<td>50</td>
</tr>
<tr>
<td>H14</td>
<td>49</td>
</tr>
<tr>
<td>H15</td>
<td>52</td>
</tr>
<tr>
<td>H16</td>
<td>51</td>
</tr>
<tr>
<td>H17</td>
<td>55</td>
</tr>
<tr>
<td>H18</td>
<td>53</td>
</tr>
<tr>
<td>H19</td>
<td>58</td>
</tr>
<tr>
<td>H20</td>
<td>57</td>
</tr>
<tr>
<td>H21</td>
<td>61</td>
</tr>
<tr>
<td>H22</td>
<td>60</td>
</tr>
<tr>
<td>H23</td>
<td>63</td>
</tr>
<tr>
<td>H24</td>
<td>62</td>
</tr>
<tr>
<td>H25</td>
<td>65</td>
</tr>
<tr>
<td>H26</td>
<td>64</td>
</tr>
<tr>
<td>H27</td>
<td>69</td>
</tr>
<tr>
<td>H28</td>
<td>66</td>
</tr>
<tr>
<td>H29</td>
<td>74</td>
</tr>
<tr>
<td>H30</td>
<td>70</td>
</tr>
<tr>
<td>H31</td>
<td>76</td>
</tr>
<tr>
<td>H32</td>
<td>75</td>
</tr>
<tr>
<td>H33</td>
<td>80</td>
</tr>
<tr>
<td>H34</td>
<td>77</td>
</tr>
<tr>
<td>H35</td>
<td>82</td>
</tr>
<tr>
<td>H36</td>
<td>81</td>
</tr>
<tr>
<td>H37</td>
<td>84</td>
</tr>
<tr>
<td>H38</td>
<td>83</td>
</tr>
<tr>
<td>H39</td>
<td>GND</td>
</tr>
<tr>
<td>H40</td>
<td>GND</td>
</tr>
</tbody>
</table>
2.9 Clock circuitry

The HAHA Board includes a 50 MHz clock signal for the FPGA and an 8 MHz clock signal for the Micro controller.

The clock that connected to the FPGA is used for clocking the user logic. In addition, the clock input is connected to the Phase Lock Loops (PLL) clock input pin of the FPGA. The user can use the clock as a source clock for the PLL circuit.

As mentioned in 2.5 Chip interconnections, there is a chip interconnection that can also be used as the clock source for the FPGA. User needs to configure the Micro controller so that IO pin will output clock signal for the FPGA. In this way, these two chips can be synchronized to the same clock.

There are more IO pins on the FPGA that can be used as clock input pin. Some of the are connected to the expansion header. User can connect them to external clock signals if needed.

All the pins that can be used as external clock inputs are listed in Table 8.
Table 8 FPGA clock input pins and their circuitry connections

<table>
<thead>
<tr>
<th>FPGA Clock Input Pin</th>
<th>FPGA Pin No.</th>
<th>Signal Name</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK0n</td>
<td>25</td>
<td>S5</td>
<td>IO (Switch)</td>
</tr>
<tr>
<td>CLK0p</td>
<td>26</td>
<td>S4</td>
<td>IO (Switch)</td>
</tr>
<tr>
<td>CLK1n</td>
<td>27</td>
<td>S3</td>
<td>IO (Switch)</td>
</tr>
<tr>
<td>CLK1p</td>
<td>28</td>
<td>CLK_inter</td>
<td>Chip interconnection</td>
</tr>
<tr>
<td>CLK6n</td>
<td>51</td>
<td>H16</td>
<td>Expansion header pin</td>
</tr>
<tr>
<td>CLK6p</td>
<td>52</td>
<td>H15</td>
<td>Expansion header pin</td>
</tr>
<tr>
<td>CLK7n</td>
<td>53</td>
<td>H18</td>
<td>Expansion header pin</td>
</tr>
<tr>
<td>CLK7p</td>
<td>55</td>
<td>H17</td>
<td>Expansion header pin</td>
</tr>
<tr>
<td>CLK2n</td>
<td>88</td>
<td>CLK_50M</td>
<td>50 MHz clock source</td>
</tr>
<tr>
<td>CLK2p</td>
<td>89</td>
<td>HT1</td>
<td>Expansion header pin</td>
</tr>
<tr>
<td>CLK3n</td>
<td>90</td>
<td>HT2</td>
<td>Expansion header pin</td>
</tr>
<tr>
<td>CLK3p</td>
<td>91</td>
<td>HT3</td>
<td>Expansion header pin</td>
</tr>
</tbody>
</table>

2.10 Using SPI devices

There are three SPI devices that can work as an SPI slave controlled by the Micro controller: an EEPROM, an accelerometer, and the Bluetooth processor. There is also an SPI slot on the HAHA Board. Users can connect more devices to the Micro controller through the slot.

Figure 8 shows the setup used with SPI on HAHA Board. (A word of caution about the SPI pin names. MISO, MOSI, SCK and SS' are the names used by AVRs. Other devices may use a different set of names. You must check the data sheet of the particular device you are using to get them right.)
In this SPI setup, the Micro controller has three slaves at the same time. Users can select which slave to communicate to by asserting the SS’ (Slave Select) pin of the slave devices. The IO pins used for slave selecting is shown in Figure 8. When selecting one of the slaves to communicate, the SS’ signal of that slave device must be asserted by setting it Low. In order to avoid confliction, SS’ for other devices must be set to High. The Micro controller can only communicate to one SPI device at a time.

When using the SPI devices, please refer their datasheets. Carefully read and write data from the correct addresses. Writing to some reserved addresses of the SPI devices will cause permanent damage to the device. Some addresses that contain the factory calibration values should not be changed.

Figure 9 shows the SPI slot pin functions. The user can expand more SPI devices by connecting corresponding pins to the slot. The user can also use it as a place to probe signals on the board.
Chapter 3 Examples for Programming

This chapter provides a number of examples of circuits and code implemented on the HAHA Board. These examples provide demonstrations of some basic features on the board, such as user LEDs and switches. For each demonstration, the resource code (Verilog code and Assembly code) is provided.

3.1 Programming the FPGA

3.1.1 Counter implemented in the FPGA

This example gives a demonstration of implementing a simple circuit into the FPGA: a 4-bit up counter. The module will need 3 inputs and 1 output. One input is the clock, one input is to reset the output to be 0000, and the third is to enable counting. The output is a 4-bit output representing the value counted. Therefore, 1 clock source and 2 switches and 4 LEDs will be used. Below are the steps.

1. Create a new project in Quartus. Select the right device that the HAHA Board uses, which is 10M50SCE144C8G as shown in Figure 10.

![Family & Device Settings](image)

**Figure 10** Select the device on the HAHA Board.

2. Create a new Verilog HDL file and add it to the project. Create the up_counter module in the v-file. Type in the code given below. The given code contains two modules: the up_counter module and a sub-module which is a frequency divider. As the clock source this example will be
using is the 50 MHz which is impossible for human eyes to discern, the divider will make the frequency much slower. After type in the code, Analysis and Elaboration the code.

Verilog Code

```verilog
module up_counter(clk, reset, enable, out);
  input clk; //clock input for the counter: 50MHz
  input reset; //reset the output to be 0000
  input enable; //enable signal
  output reg [3:0] out; //4-bit output
  wire clk_slow; //a divided clock

  always @(posedge clk) begin
    if (reset) out=4'b0000;
    else if (enable) out=out+4'b0001;
  end

  f_divider U0 (.clk(clk), .clkout(clk_slow)); //sub module frequency divider instantiated
endmodule

/* Below is a sub module frequency divider to make the clock frequency slower, so that LED output can be seen clearly. */

module f_divider(clk, clkout);
  input clk; //clock input
  output reg clkout; //divided clock output
  reg [25:0] cnt; //26-bit register

  always @(posedge clk) begin
    cnt=cnt+1;
    if (cnt==26'b0011111010111001000010000000) begin
      clkout=~clkout;
      cnt=26'b0;
    end
  end
endmodule
```

3. Assign signals to the right pins. Open the pin planner window as shown in Figure 11. Give the all the signals the right location according to this HAHA user manual.
4. Start compilation. When finished, the SOF-file will be created.
5. Open the programmer window. Use the SOF-file to program the FPGA.
3.1.2 Arithmetic and Logic Unit

This example shows a simple Arithmetic and Logic Unit (ALU). The ALU has two 4-bit inputs \( a \) and \( b \), and a one 2-bit function selection input \( \text{sel} \). The ALU will do different operations according to different values of the function selection input.

<table>
<thead>
<tr>
<th>Selection bits</th>
<th>Function (o)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>( a + b )</td>
</tr>
<tr>
<td>01</td>
<td>( a \times b ) (only the least 4 bits)</td>
</tr>
<tr>
<td>10</td>
<td>( a \text{ and } b ) (bitwise)</td>
</tr>
<tr>
<td>11</td>
<td>( a \text{ OR } b ) (bitwise)</td>
</tr>
</tbody>
</table>

Input ports can be used to provide the operands and selections bits. Switches and buttons on the HAHA Board can be used. The output can be shown on four LEDs.

Using Verilog, this ALU can be realized in several ways. Such as using ‘case’, the code will be look like below.

```verilog
module alu(a, b, sel, o);
    input [3:0] a, b;
    input [1:0] sel;
    output reg [3:0] o;
    always @(a or b) begin
        case (sel)
            2'b00: o = a + b;
            2'b01: o = a * b;
        default: o = 0;
        endcase
    end
endmodule
```
Verilog Code

module parking_lot_controller (clk, rstn, sense0, sense1, sign_full, sign_free);

    // maximum number of cars in the lot at once
    parameter max_cars = 10;

    // sense0 and sense1 are the two light sensors: sense0 is "outside" sensor (closer to outside)
    // and sense1 is "inside" sensor (closer to inside)

endmodule

After compilation, use pin planner to assign the I/O ports to suitable pins as shown in Figure 12.

<table>
<thead>
<tr>
<th>Node Name</th>
<th>Direction</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[3]</td>
<td>Input</td>
<td>PIN_21</td>
</tr>
<tr>
<td>a[2]</td>
<td>Input</td>
<td>PIN_22</td>
</tr>
<tr>
<td>a[1]</td>
<td>Input</td>
<td>PIN_23</td>
</tr>
<tr>
<td>a[0]</td>
<td>Input</td>
<td>PIN_25</td>
</tr>
<tr>
<td>b[3]</td>
<td>Input</td>
<td>PIN_26</td>
</tr>
<tr>
<td>b[2]</td>
<td>Input</td>
<td>PIN_27</td>
</tr>
<tr>
<td>b[1]</td>
<td>Input</td>
<td>PIN_29</td>
</tr>
<tr>
<td>b[0]</td>
<td>Input</td>
<td>PIN_30</td>
</tr>
<tr>
<td>o[3]</td>
<td>Output</td>
<td>PIN_141</td>
</tr>
<tr>
<td>o[2]</td>
<td>Output</td>
<td>PIN_140</td>
</tr>
<tr>
<td>o[1]</td>
<td>Output</td>
<td>PIN_135</td>
</tr>
<tr>
<td>o[0]</td>
<td>Output</td>
<td>PIN_133</td>
</tr>
<tr>
<td>se[1]</td>
<td>Input</td>
<td>PIN_85</td>
</tr>
<tr>
<td>se[0]</td>
<td>Input</td>
<td>PIN_86</td>
</tr>
</tbody>
</table>

Figure 12 ALU pin plan.

Then, generate the bitstream file and program the FPGA to see if its function works well.

3.1.3 Finite State Machine

A finite-state machine (FSM) or simply a state machine is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of states.

One of the UF green parking lots has only one entrance gate and can accommodate only 10 cars. Only one car can pass through the entrance gate (either exit or enter at a time). Two light sensors, separated by 1 meter, detect whether the car is entering or leaving. During entrance, there will be a sign with FULL or FREE.
// we assume they are active low, e.g. "on" when the light beam is broken and output is 0

input clk, rstn, sense0, sense1;
output sign_full, sign_free;

integer cars_in_lot;
reg [3:0] state;

// we assume that to enter, sense0 is broken first, followed by both sense0 and sense1
// as they are only 1 meter apart, and finally only sense1.

assign sign_full = (cars_in_lot >= max_cars)? 1'b1: 1'b0;
assign sign_free = ~sign_full;

always @ (posedge clk or negedge rstn) begin
  if (rstn == 1'b0) begin
    cars_in_lot = 0;
    state = 4'b0000;
  end
  else begin
    // this state should
    if (state == 4'b0000) begin
      if (sense0 == 1'b0 && sense1 == 1'b1) begin // incoming cars
        state = 4'b0001;
      end
      else if (sense0 == 1'b1 && sense1 == 1'b0) begin // exiting cars
        state = 4'b0100;
      end
      else begin // continue to wait for a car
        state = 4'b0000;
      end
    end
    else if (state == 4'b0001) begin // incoming car
      // car is still crossing outside sensor
      if (sense0 == 1'b0 && sense1 == 1'b1) begin // incoming cars
        state = 4'b0001;
      end
      else if (sense0 == 1'b1 && sense1 == 1'b0) begin // exiting cars
        state = 4'b0100;
      end
      else begin // continue to wait for a car
        state = 4'b0000;
      end
    end
    else if (state == 4'b0000) begin // incoming car
      // car is still crossing outside sensor
      if (sense0 == 1'b0 && sense1 == 1'b1) begin // incoming cars
        state = 4'b0001;
      end
      else if (sense0 == 1'b1 && sense1 == 1'b0) begin // exiting cars
        state = 4'b0100;
      end
      else begin // something else happened (e.g. car has just pulled in, but
      // left before entering)
        state = 4'b0000; // return to init state without
        changing count
      end
    end
  end
end
end

else if (state == 4'b0010) begin  // car crossed both sensors on its way in

    // car is still crossing threshold
    if (sense0 == 1'b0 && sense1 == 1'b0) begin
        state = 4'b0010;
    end
    // car has finished crossing outside sensor, and is passing inside sensor
    else if (sense0 == 1'b1 && sense1 == 1'b0) begin
        state = 4'b0011;
    end
    // something else happened (e.g. car left without entering)
    else begin
        state = 4'b0000;
    end
end

else if (state == 4'b0011) begin  // car has finally entered; increment count and go to initial state

cars_in_lot = cars_in_lot + 1;
state = 4'b0000;
$display("Car has entered lot (%2d total).", cars_in_lot);
end

else if (state == 4'b0100) begin  // exiting car

    // car is still crossing inside sensor
    if (sense0 == 1'b1 && sense1 == 1'b0) begin
        state = 4'b0100;
    end
    // car has crossed both sensors
    else if (sense0 == 1'b0 && sense1 == 1'b0) begin
        state = 4'b1000;
    end
    // something else happened (e.g. car has just pulled in, but left before entering)
    else begin
        state = 4'b0000;  // return to init state without changing count
    end
end

else if (state == 4'b1000) begin  // car is still crossing threshold

    // car has finished crossing inside sensor, and is passing outside sensor
    if (sense0 == 1'b0 && sense1 == 1'b0) begin
        state = 4'b1000;
    end
end

else if (state == 4'b1001) begin  // car has crossed both sensors on its way in

    // car is still crossing threshold
    if (sense0 == 1'b0 && sense1 == 1'b0) begin
        state = 4'b1001;
    end
end

else if (state == 4'b0101) begin  // entering car

    // car is still crossing outside sensor
    if (sense0 == 1'b0 && sense1 == 1'b0) begin
        state = 4'b0101;
    end
end

else begin

    state = 4'b0000;  // return to init state without changing count
end
state = 4'b1100;
end
// something else happened (e.g. car left without entering)
else begin
    state = 4'b0000;
end
end

else if (state == 4'b1100) begin // car has finally exited; decrement
cars_in_lot = cars_in_lot - 1;
state = 4'b0000;
$display("Car has exited lot (%2d total.", cars_in_lot);
end
else begin // otherwise something went wrong - go back to init

state = 4'b0000;
end
end

endmodule

State Diagram:

Go to Tools → Netlist Viewer → State Machine Viewer to verify if the designed FSM fulfills the specifications.

Simulation:

Go to New → University Program VWF → Edit → Insert → Insert Node or Bus → Node Finder → list → Select all → Insert values for the inputs → Simulation.
3.1.4 Read-write operation of an SRAM

The following Verilog code can be used to read (write) SRAM data from (to) an SRAM. This SRAM reads and writes 8-bit data.

```
module SRAM_top(clk, addr, read_write, clear, data_in, data_out);

parameter n = 4;
parameter w = 8;

input clk, read_write, clear;
input [n-1:0] addr;
input [w-1:0] data_in;
output reg [w-1:0] data_out;

// Start module here!
```

Verilog Code for SRAM read-write operation
3.2 Programming the Micro controller

3.2.1 Counter programmed in the Micro controller

This example gives a demonstration of programming the simple counter into the Micro controller. The function will be the same with the circuit in 3.1. The difference is that this one will be realized by the Micro controller. As on the HAHA Board, the Micro controller is not connected to many switches and LEDs due to its IO pin number limitation, the Micro controller will get access to these peripherals through the FPGA which will work as an interface connecting them up. Below are the steps.

1. Open the Atmel Studio software and create a new AVR assembler project. Select the right device, which is ATmega16U4.
2. Type in code in the ASM-file.

```assembly
.include "m16u4def.inc"
.org 0
rjmp main

main:
; configure PD0 to PD3 to be output pins and configure PD4 to PD7 to be input pins:
ldi r16, 0b00001111
out DDRD, r16

ldi r17, 0x00 ; reset register r17 to be 0x00
```

```vhdl
reg [w-1:0] reg_array [2**n-1:0];

integer i;
initial begin
for( i = 0; i < 2**n; i = i + 1 ) begin
  reg_array[i] <= 0;
end
end

always @(negedge(clk)) begin
  if( read_write == 1 )
    reg_array[addr] <= data_in;
  //if( clear == 1 ) begin
  //  for( i = 0; i < 2**n; i = i + 1 ) begin
  //    reg_array[i] <= 0;
  //  end
  //end
  data_out = reg_array[addr];
end
endmodule
```
loop:
    sbic PIND, 4 ;skip the next line if enable not asserted
    inc r17 ;increase the value of r17 by 1
    sbrc r17,4 ;skip the next line if value isn't bigger than 00010000
    ldi r17, 0x00 ;clear the counter value
    sbis PIND, 5 ;skip the next line if reset not asserted
    ldi r17, 0x00 ;reset the value to be 0
    out PORTD, r17 ;output the value to the IO ports
    call delay ; slow down so human eyes can see clearly
    rjmp loop

;all below is to delay the program by letting it count
delay:
    ldi r23, 0x00
    delay_inc:
        call delay1
        inc r23
        sbrs r23, 7
        rjmp delay_inc
        ret
    delay1:
    ldi r24, 0x00
    delay1_inc:
        inc r24
        sbrs r24, 7
        rjmp delay1_inc
        ret

3. Build the project. When finished, a HEX-file will be created.
4. Program the Micro controller with HEX-file as shown in Figure 13.
5. Power OFF and On the HAHA Board. Program the FPGA so that 4 LEDs are directly connected to the Micro controller’s IO pins PD0 to PD3, 2 switches are directly connected to the Micro controller’s IO pins PD4 and PD5. This will require repeating the steps in 3.1 but with different code.

3.2.2 Light up a LED through an I/O port

When using an I/O port, user have to configure the pin first. Each port pin consists of three register bits: DDxn, PORTxn and PINxn. The DDxn bit in the DDRx register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.

From the HAHA Board, it can be found there is a red LED connected to PORTB7. Therefore, configure that port as output first by code:

```c
sbi DDRB, 7
```

Then, make the 7th bit of PORTB to be 1 to produce a high level on that port:

```c
sbi PORTB, 7
```

At last, finish the program by entering an endless loop:

```c
Loop: rjmp loop
```
The code of this easy program is given below.

```assembly
.include "m16u4def.inc"
.org 0
rjmp main

main:
    sbi DDRB, 7
    sbi PORTB, 7

loop:
    rjmp loop
```

### 3.2.3 Make the LED to blink

The LED will blink when the I/O port toggles between 1 and 0. The I/O can output a low level by writing the code:

```assembly
cbi PORTB, 7
```

However, people cannot see the LED blink if the LED blink at the frequency of the chip internal clock. Therefore, a delay can be inserted to slow the process down.

A delay function can be created by making a working register count in a loop, and the program can only jump out of the loop when the register counts to a certain value. The delay function can be written as:

```assembly
delay:
    ldi r23, 0x00
delay_inc:
    inc r23
    sbrs r23, 7
    rjmp delay_inc
ret
```

At the beginning of the delay function, the value of the working register r23 is initiated to 0x00 by using ldi instruction. Then its values is increased by one by using inc instruction. The program will jump back to the label delay_inc until the 7th bit of r23 is set to 1. Here the function of sbrs is to check the value of the 7th bit of register r23. The the bit value is 1, then the following instruction will be skipped.

The code of the whole program is given below. (Actually, the delay time provided in this program is still not long enough for human eyes to see clearly. Users can make the delay longer by calling another delay in the delay loop.)

```assembly
.include "m16u4def.inc"
.org 0
rjmp main

main:
    ldi r16, 0xFF
    out DDRB, r16

loop:
    sbi PORTB, 7
```
3.2.4 Read a value from a SRAM address and send it to I/O ports

There is a SRAM inside the micro controller. Users can read a value from a SRAM address to a working register by doing:

```
lds r17, 300; 300 is an SRAM address
```

Configure all the bits of PORTD to be output port:

```
ldi r16, 0xFF
out DDRD, r16
```

Send the value from a working register to the ports by using:

```
out PORTD, r16
```

Therefore, the whole program will be like what is given below.

```
.include "m16u4def.inc"
.org 0
rjmp main

main:
  ldi r16, 0xFF
  out DDRD, r16
  lds r17, 300
  out PORTD, r17
loop:
  rjmp loop
```

3.2.5 Program controlled by an input port

Sometimes a program will need signal read from the I/O port which is configure as an input port. For example, a program is waiting for a high level from an input port to go on execution. If the signal come into the port is always 0, then the program will be stuck.

```
.include "m16u4def.inc"
.org 0
rjmp main

main:
  cbi DDRC, 7
```
3.2.6 IO value to control a LED

User can propagate a value from an input port to an output port. For example, an input port is connected to a switch and an output port is connected to a LED. In this way, the switch can control the LED by propagate its value to the LED.

Assembly Code

```
.include "m16u4def.inc"
.org 0
rjmp main

main:
    ;set pc7 as input
    cbi DDRC, 7

    ;set pb7 as input
    sbi DDRB, 7

loop:
    ;do not set pb7 if pc7 is clear
    sbic PINC, 7
    sbi PORTB, 7

    ;do not clear pb7 if pc7 is set
    sbis PINC, 7
    cbi PORTB, 7

rjmp loop
```

3.2.7 Simple calculation example

There are arithmetic and logic instructions in the instruction set. Arithmetic instructions include ADD, SUB, MUL, and so on. This example will show how to do a simple function using several working registers. For example, the function is f(x)=255-2x, if x<128; and otherwise f(x)=0.

Instructions needed here include MUL, SUB and a branch instruction. The calculation result will be output to portD. Code is provided below.

Assembly Code

```
.include "m16u4def.inc"
.org 0
rjmp main

main:
    ;set PortD as output
    ldi r16, 0xff
    out DDRD, r16

    ;store x=100 in r18 and store 2 in r19
    ldi r18, 0x64
    ldi r19, 0x02
```
;if x is 128 or bigger, jump to big_option
sbrc r18, 7
rjmp big_option

;calculate 2*100 and store the result in r18
mul r18, r19
mov r18, r0

;calculate 255-r18 and store the result in r18
ldi r19, 0xff
sub r19, r18
mov r18, r19
rjmp loop

big_option:

ldi r18, 0x00

loop:
;output the final result
out PORTD, r18
rjmp loop
Appendix

Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Change Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1.0</td>
<td>Initial Version</td>
</tr>
<tr>
<td>V1.1</td>
<td>Update FPGA to 10M04SCE144C8G</td>
</tr>
<tr>
<td>V1.2</td>
<td>Update FPGA to 10M50SCE144C8G</td>
</tr>
</tbody>
</table>

Copyright Statement

Copyright © 2016 University of Florida. All rights reserved.