APPENDIX

THE HARDWARE HACKING (HaHa) PLATFORM FOR HANDS-ON TRAINING

A

In this Appendix, we provide details of the custom hardware module, HaHa, which is designed by the authors to perform all the hands-on experiments in a single platform. The experiments described in the book are listed in Table A.1. These experiments are designed to provide valuable practical experience on various security issues at the chip and PCB level and countermeasures for some of the major threats. More details about the experiments are available in the supplementary document. Please visit: *http://hwsecuritybook.org*.

Table A.1 Hands-on experiments described in the book

	Experiment	Expected Learning Outcome	Chapter
1	Reverse engineering attacks	*The vulnerability of PCBs to RE and cloning attacks*	4
2	Hardware Trojan attacks	*Different forms of Trojan attacks with varying triggers/payloads, and possible protection mechanisms*	5
3	Power side-channel attacks	*Leakage of the secret key from a crypto module through supply current measurement and analysis of data*	8
4	Fault injection attacks	*Leakage of the secret key from a crypto module through injection of fault*	8
5	JTAG-based attacks in PCB	*Information leakage or functional manipulation of PCB using the JTAG interface*	9
6	Bus snooping attacks	*Information retrieval from a PCB through physical access (probing of metal traces)*	11
7	Design of security primitives: (a) Physical unclonable function, (b) True random number generator	*Design philosophy and quality assessment of major security primitives*	12
8	Hardware obfuscation: (a) combinational locking, (b) finite state machine based obfuscation	*Techniques for functional and structural transformation of a design to prevent RE and evaluation of the techniques*	14
9	PCB tampering attacks (Modchip)	*Physical manipulation of PCB traces to alter their functionality*	15
10	SoC security policy	*SoC security architecture and protection for the assets*	16

The experiments use different features and configurable options provided in the HaHa platform. Students can work on this platform to understand the working principle and structure of a computer system, and ethically "hack" it at different levels. They can examine it to understand various security

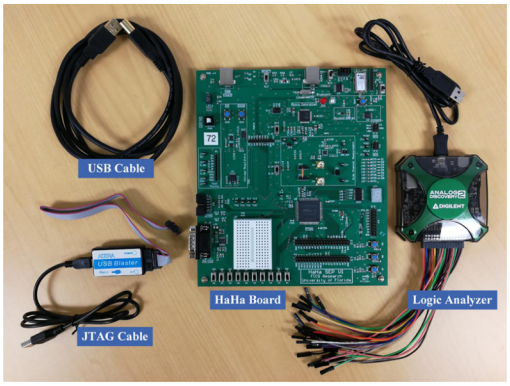


FIGURE A.1

The HaHa kit provides a unique self-contained platform for hardware security training and education. The kit is suitable for online offering of a hardware security lab course, where the students can acquire the kit from various possible sources, and perform all the experiments at home without the need for a physical lab, or special benchtop equipments.

vulnerabilities, mount attacks, and implement countermeasures. Figure A.1 depicts the HaHa board, a JTAG programming cable, a USB connector, an oscilloscope, and logic analyzer (example, Analog Discovery 2 from Digilent Inc.), which together make the HaHa kit self-contained and suitable for performing all these experiments at home without the need for any benchtop equipment, such as an oscilloscope, power supply, or waveform generator. The HaHa board needs to be interfaced with a computer's USB port. Two software modules, as described later in this Appendix, need to be installed on the computer. The HaHa kit provides a unique take-home lab for learning innards of a computer system, different aspects of hardware security issues, and associated countermeasures. Figure A.2 illustrates how these components can be connected with a computer to create the experimental platform.

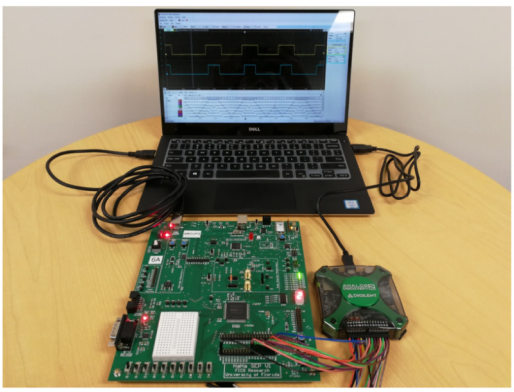


FIGURE A.2

The experimental setup using the components of the HaHa kit. The HaHa board needs to be connected to a computer through its USB port for most of the experiments.

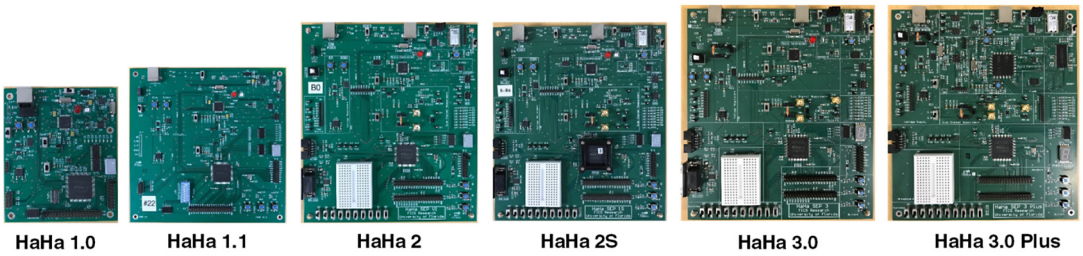


FIGURE A.3

Design of the HaHa board has been revised over the years to improve its capability to run diverse hardware security experiments. The figure shows three generations of HaHa board and their relative sizes.

Figure A.3 shows the evolution of HaHa board over several years. It has been updated over three generations to improve students' learning experience and to augment its flexibility to run different hardware security experiments in one platform. The later is not possible with commercially available FPGA or processor development boards, or special-purpose boards, such as the Sakura side-channel attack evaluation board (http://satoh.cs.uec.ac.jp/SAKURA/link.html). These existing boards are suitable for implementing one or few experiments (for example, Trojan attacks or security primitive design in Altera/Xilinx FPGA development boards, or power side-channel attack in Sakura board), but are not amenable for implementing many others.

Students at University of Florida and Case Western Reserve University have got access to the HaHa board for the past five years in their hardware security lab courses. Figure A.4 shows students using different versions of the board to implement different parts of a computing system (for example a smart

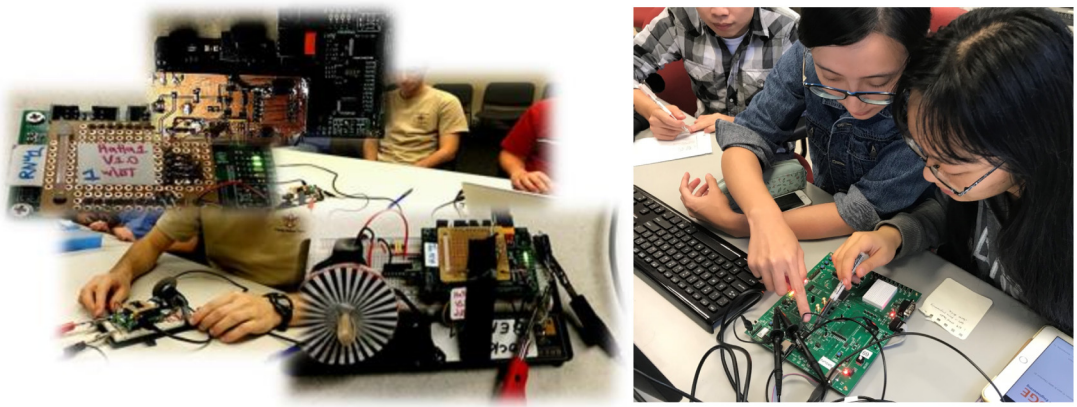


FIGURE A.4

Students in Case Western Reserve University and University of Florida work with different versions of the HaHa kit to implement models of computer systems and perform various hardware security experiments.

fan or pedometer) and then mounting diverse hardware attacks (such as, chip-/PCB-level hardware attacks), and countermeasures. The development of the board has been supported in part by United States National Science Foundation (NSF) through a grant.

The book website provides a companion material with detailed lab instructions for students to perform these experiments, make necessary observations, and write reports. Most of the experiments include advanced options, which are appropriate for students who like to explore further and are interested to perform more complex attacks and implement sophisticated countermeasures. The platform is designed with the flexibility for implementing simple models of various computing systems. Table A.2 lists the key features of the HaHa board. It includes both an FPGA (Altera MAX 10 series FPGA with embedded temperature sensor) and a microcontroller/processor (ARM or Atmel, depending on the model), along with a nonvolatile memory (EEPROM or flash), accelerometer, programmable voltage source, current sensing mechanism, JTAG connection between the FPGA and the microcontroller, instrumentation amplifier to improve signal-to-noise (SNR) ratio for side-channel measurements, and an optional socket for PCB to support experiments, such as Trojan detection and security primitive design, which involve measurements over multiple FPGA chips.

Table A.2 Select list of features of the HaHa board

Number	Feature description	Benefit for security experiments
1	The board includes both FPGA and microcontroller chips	Helps to implement diverse types of a computing systems using either of the components or both
2	FPGA and microcontroller both have embedded flash and are easily programmable via JTAG	Both devices are easy to program; FPGA can act as an accelerator for the microcontroller
3	FPGA and microcontroller communicate with the nonvolatile memory	Helps to build an independent computing system with its own memory to store configuration and input data
4	Bidirectional connection between FPGA and microcontroller	Offers various system configurations with any of them acting as the CPU, for example, microcontroller as CPU and FPGA as an accelerator
5	Connection to all peripherals through both the FPGA and the microcontroller	Offers flexibility to access any peripheral through either FPGA or microcontroller
6	Current/power monitoring for FPGA	Performs side-channel analysis and attack experiments
7	Current/power monitoring for the entire board (only available in the latest version of the HaHa board)	Offers side-channel analysis for the board and PCB authentication
8	Voltage control of FPGA using a potentiometer	Offers robustness analysis capabilities for PUF/TRNG experiments and fault injection attacks with power glitch
9	Voltage control of microcontroller using a potentiometer	Offers robustness analysis capabilities for PUF/TRNG experiments and fault injection attacks in embedded software with power glitch
10	JTAG chain for FPGA and microcontroller	Performs JTAG programming, JTAG Testing, and JTAG attack experiments
11	Two-layer board	Facilitates reverse engineering through visual inspection

(continued on next page)

Table A.2 (*continued*)

Number	Feature description	Benefit for security experiments
12	Simple layout with clearly marked regions	Provides a clear understanding of the board design and configurations, and helps to perform reverse engineering attack
13	Embedded breadboard and user headers	Allows an user to include additional components and small custom circuits in the board
14	Configurable Bluetooth module	Offers Bluetooth connection between boards to build a connected system and Bluetooth attack experiment
15	Embedded temperature sensor in the FPGA	Enables temperature-triggered Trojan attack experiment, PUF/TRNG reliability analysis and side-channel experiments
16	Plenty of configurable headers	Enables physical tampering (Modchip) attack experiment
17	Clock source and PLL inside the FPGA	Fault injection attack using the clock
18	Integration of expansion headers	Offers flexibility to add more components or sister boards to HaHa to expand its functionality
19	Multiple probing points on the buses	Enables bus snooping attack and PCB tampering experiments

Next, we describe the design of the HaHa board (Version 2) in detail and provide the schematic, layout, and bill-of-materials (BoM), so that readers who are interested to create their own HaHa kit, which requires fabrication, assembly, and testing of the custom board, can accomplish the task. Additionally, the whole kit, and its components can be individually purchased from third-party vendors. While current edition of the book includes 10 experiments as listed in Table A.1, the authors intend to add more experiments using the HaHa kit as a companion material in the future, which will be available through the book website. These new experiments are expected to cover system-level security issues and solutions, such as the timing side-channel attack, EM analysis attack, more complex forms of Trojan attacks, and additional security primitives, such as aging sensors for counterfeit IC detection. The authors expect to add these new experiments to the online companion materials within a year from the publication of this book.

A.1 HaHa BOARD

This section describes the features and design characteristics of the HaHa board in detail.

A.1.1 LAYOUT AND COMPONENTS

A photograph of the HaHa board is shown in Fig. A.5, which illustrates the configuration of the board and highlights the locations of all connectors and key components.

The following hardware components are included in the HaHa board:

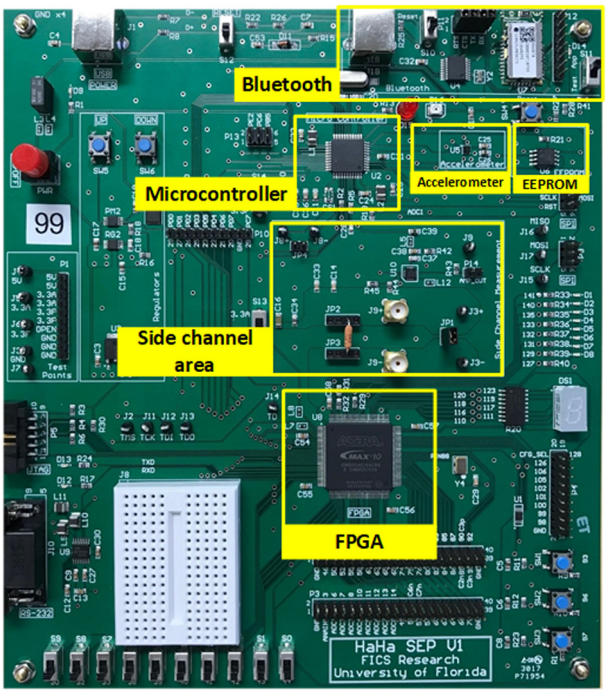


FIGURE A.5

The HaHa board (Version 2) showing the major components.

- Intel Altera MAX 10 FPGA device (with 50,000 logic elements)
- Atmel AVR 8-bit microcontroller
- JTAG header for programming the FPGA and the microcontroller
- SPI (Serial Peripheral Interface) expansion headers
- USB Type B port for programming the microcontroller
- High-performance 3-axis accelerometer
- 1 Mbit serial EEPROM memory
- Bluetooth specification v4.0 compliant Bluetooth network processor
- Adjustable regulator providing variable power source from 1.5V to 3.6V
- Two toggle switches for selecting the power source from the fixed 3.3V or the adjustable source
- 1 Ohm current sensing resistor and instrumentation amplifier
- 3 pushbutton switches
- 10 slide switches
- 8 user LEDs
- 50-MHz and an 8-MHz oscillator for clock sources
- 3 I/O pin expansion headers
- 7 segment display
- Photodiode

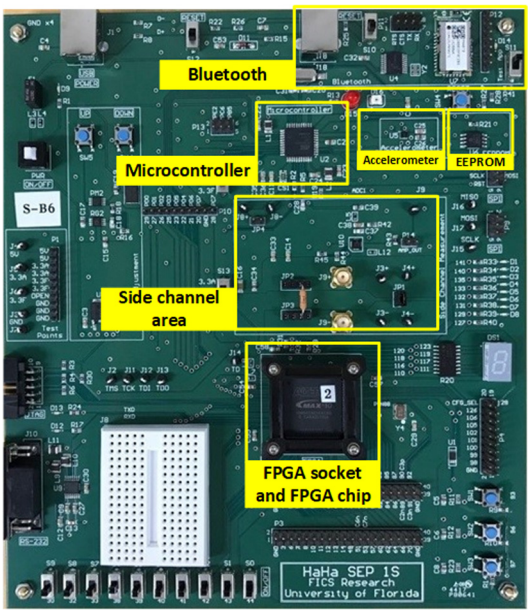


FIGURE A.6

Version of the HaHa board with a socket for the FPGA chip.

An alternative version of the HaHa board (shown in Fig. A.6) is developed with a socket for the FPGA. The socket is for the 144-pin Intel Altera MAX 10 FPGA, and the chip can be inserted or taken off the socket very easily. The socket is surface-mounted on the board with all other components and configurations remaining unchanged. With this version of the board, users can perform certain experiments that require multiple FPGA chips, for example, Trojan detection or the security primitive (PUF/TRNG) design experiments. These experiments often require multiple measurements of some physical parameters (such as, static or dynamic supply current, signal propagation delay) over a number of chips to evaluate the impact of process variations, environmental parameter (such as, power supply noise or temperature) variations, and aging.

A.1.2 BLOCK DIAGRAM OF THE HaHa BOARD

Figure A.7 shows the block diagram of HaHa board. Two major components of the board are the FPGA and the microcontroller. These chips are interconnected with the JTAG chain. They can be programmed and tested using the JTAG interface. Additionally, they are connected through a 9-bit bus. One of the bus lines can be used as the clock. In that case, it becomes an 8-bit bus with a clock.

Figure A.7 also shows the chips' peripheral functions. Multiple serial peripheral interface (SPI) devices can be connected to the microcontroller using the SPI interface. The SPI devices include an accelerometer, a nonvolatile memory (EEPROM or flash), and a Bluetooth module. Two additional

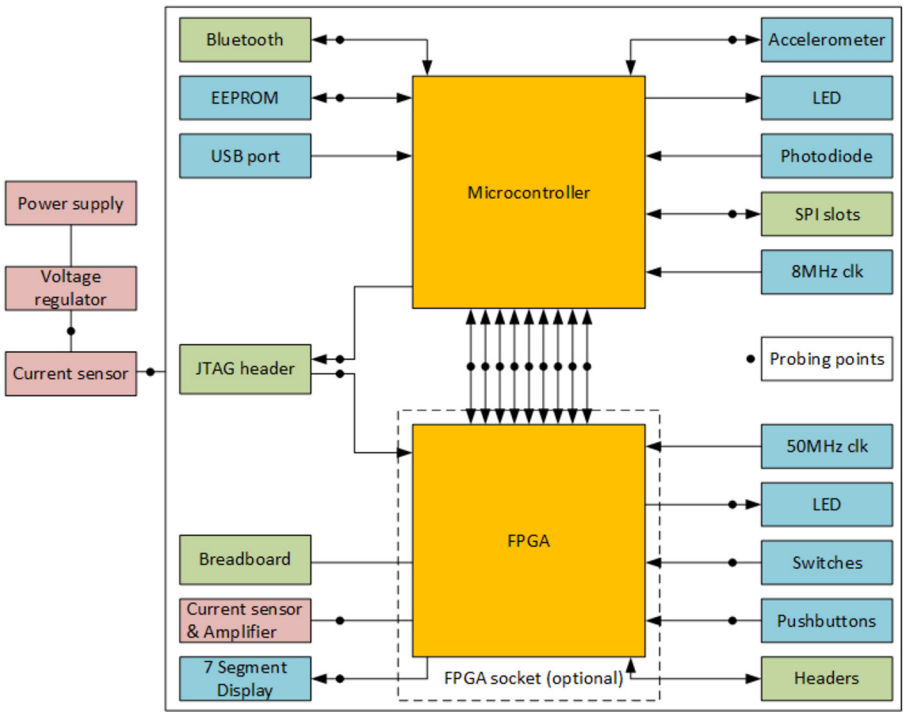


FIGURE A.7

Block diagram of the HaHa board.

SPI slots are provided that allow more chips, or sister boards with SPI interface to be connected to the board.

The FPGA, which comes with more I/O pins than the microcontroller, enables connections to a larger number peripheral devices. These peripheral devices include slide switches, push buttons, LEDs, a 7-segment display, and user expansion headers.

All of the peripherals are accessible by both chips. For example, when the FPGA wants to communicate with the on-board accelerometer, it can use the microcontroller as the medium to talk to the accelerometer. If a user wants to control the microcontroller with a switch, the FPGA can be programmed to work as the medium. Therefore, both chips can talk to any peripheral device on the board, and hence, act as the central processing unit (CPU) of a computing system that can be built on the HaHa board.

The board has well-defined configurable user expansion ports, which make it very flexible to meet a user's needs. It is equipped with three user expansion headers that are connected to the FPGA, two SPI slots that are connected to the SPI interface of the microcontroller, and a breadboard that can be used to mount other chips and implement a user's custom circuit. These headers, slots, and the breadboard use standard components, and are compatible with connectors, wires, and chips that are not included in the HaHa board.

Users can also reconfigure the ways that the chips are connected to each other. There are several accessible pins available to the users for the two main chips. In addition, other peripherals, such as the Bluetooth module, have multiple pins capable of reconfiguration. The users of the board have the freedom to change the way that the components talk to, and work with, each other. Later, we provide some examples to show how the board can be configured to perform different experiments for hardware security education and research.

Two or more HaHa boards can be connected using wire or wireless connection mechanism in various configurations to build a connected system. The headers, probing points, and breadboards make it possible to connect two boards using jumpers and wires. On the other hand, the Bluetooth module enables each HaHa board to communicate and control another. The ability to reconfigure the HaHa boards into a connected system offers a vast number of experimental options for users.

A.1.3 COMPONENTS OF HaHa

In this section, we provide information on each major component of the board.

- Altera 10M50SAE144C8G FPGA
  - 50k logic elements
  - 1638k M9k memory
  - 5888Kb user flash memory
  - 101 user I/O pins
  - 4 PLLs
  - 1 ADC with a temperature sensor
  - Boundary-scan capabilities according to the JTAG standard
- Atmel ATmega16U4-au microcontroller
  - 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 microcontroller
- SPI header
  - On-board header for programming the microcontroller
  - Offers expansion slots for add-on SPI modules
- USB B port
  - On-board USB B port for programming the microcontroller
  - 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 First-in, first-out (FIFO)
- Bluetooth processor
  - Bluetooth v4.0 compliant
  - SPI based application controller interface
  - AES security co-processor
- Adjustable regulator
  - Provides a flexible power source ranging from 1.5V to 3.6V
  - 64 steps resolution
  - 500 mA current capability
  - Two push-buttons control to increase or decrease the voltage
- 2 slide switches to select between power sources
  - Select from fixed 3.3V and adjustable source
  - When the switch is "OFF", the supply power source is fixed 3.3V
  - When the switch is "ON", the supply power source is the adjustable voltage source (1.5V to 3.6V)
- Current sensing resistor
  - 1 Ohm resistance
  - 0.5% tolerance
  - 20A maximum current
  - Instrumentation amplifier incorporated into the power supply lines across the sense resistors
  - Sense resistor placed between the decoupling capacitors and the supply pins to maintain high-frequency components in dynamic current
- Pushbutton switches
  - 3 pushbutton switches
  - Denouncing switch
- Slide switches
  - 10 slide switches
  - A switch causes logic 0 when in the ON (Left) position and logic 1 when in the OFF (Right) position
- LEDs
  - 8 LEDs connected to the FPGA
  - 1 Red LED connected to the microcontroller
  - The Red LED can be controlled by an 8-bit timer/counter with PWM (Pulse Width Modulator)
- Clock inputs
  - 8-MHz oscillator for the microcontroller
  - 50-MHz oscillator for the FPGA
  - The microcontroller shares its clock through chip interconnection
  - Both the chips have internal clocks
- Two I/O pin expansion headers
  - Connected to the I/O pins of the FPGA
  - Has 40 pins and 20 pins respectively
- 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 microcontroller
  - Peak optical wavelength 890 nm
- Breadboard
  - Double-sided tape with 170 tie-points

A.2 OPERATION INSTRUCTIONS

In this section, we provide instructions on using the HaHa board to perform various experiments.

A.2.1 POWERING UP THE HaHa BOARD

A USB Type A to B cable is included with the HaHa kit. It connects the HaHa board to computer or USB power port. One should perform the following steps to power up the board.

1. Ensure the power ON/OFF switch on the HaHa board is in the OFF position and 4 stands on the 4 corners are properly mounted. Place the board on a stable flat surface; make sure there are no conductive paths underneath the board to ground that may cause an electrical short-circuit.
2. Select the power source for the FPGA and the microcontroller. Turn OFF both selection switches to direct the power source to fixed 3.3V. Turn OFF the reset switch for the microcontroller.
3. Plug in the USB cable's Type-B port into the board and Type-A port into a computer's USB port.
4. Turn on the power ON/OFF switch. A red LED will light up to indicate the power is on.

A.2.2 INSTALL SOFTWARE MODULES IN THE COMPUTER

Altera Quartus software (edition 15 or later) needs to be installed on the computer. The Quartus Lite edition is a free version of the tool available for download from Altera's official website. You also need to install the Atmel Studio (Version 7 or later) on your computer. This software is available for free to download from the official website of Microchip.

A.2.3 CONFIGURING THE ALTERA MAX 10 FPGA

The HaHa kit includes a USB-Blaster Download Cable as shown in Fig. A.8, which can be used to program the Altera MAX 10 FPGA. One side of the USB-Blaster needs to be connected to a computer's USB Port, and the other side needs to be connected to the JTAG 10-pin header on the board.

Secure the 10-pin female connector (Fig. A.9) and the JTAG header on the HaHa board. Connect the marked pin numbers on the HaHa board accordingly. Note: the red wire in the band is for pin 1.

After connecting the USB-Blaster Download Cable into the HaHa board and a computer, locate the 2 chips in the JTAG chain on the board (the FPGA and the microcontroller). In the program window of Altera Quartus (Fig. A.10), "Auto Detect" can be used to detect two devices: the FPGA and the microcontroller. Replace the FPGA with the SOF-file that you made, then you can start programming the FPGA.

FIGURE A.8

Altera USB-Blaster Download Cable.

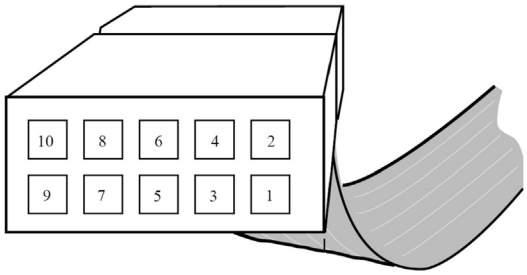


FIGURE A.9

10-pin female connector.

Note: The FPGA will lose all the configurations after it is powered off. Hence, you need to reprogram it when the board is powered-up again.

A.2.4 CONFIGURING THE MICROCONTROLLER THROUGH THE USB PORT

The Atmel AVR microcontroller ATmega16U4-au can be configured in various ways, including JTAG, SPI, and USB. The JTAG and SPI methods require a separate component, namely, the Atmel Debugger, that is not included with the HaHa board.

Note: To obtain more information about programming the AVR microcontroller using different kinds of debuggers, visit the MicroChip/Atmel website.

You can directly program the 16-KB in-system self-programmable flash through the USB port. After powering-up the HaHa board, it should detect the microcontroller in the Atmel Studio. Open the device programming window as shown in Fig. A.11. Secure the microcontroller to verify that it is connecting with the computer. The device signature can be read, and the whole chip can be erased using the interface.

Configure the programmable flash by selecting the HEX-file you generated, and clicking the program button.

Note: The content of the flash will not be erased after the power is OFF. Therefore, you can use the same function multiple times until you reset the chip by turning on the reset switch. Moreover,

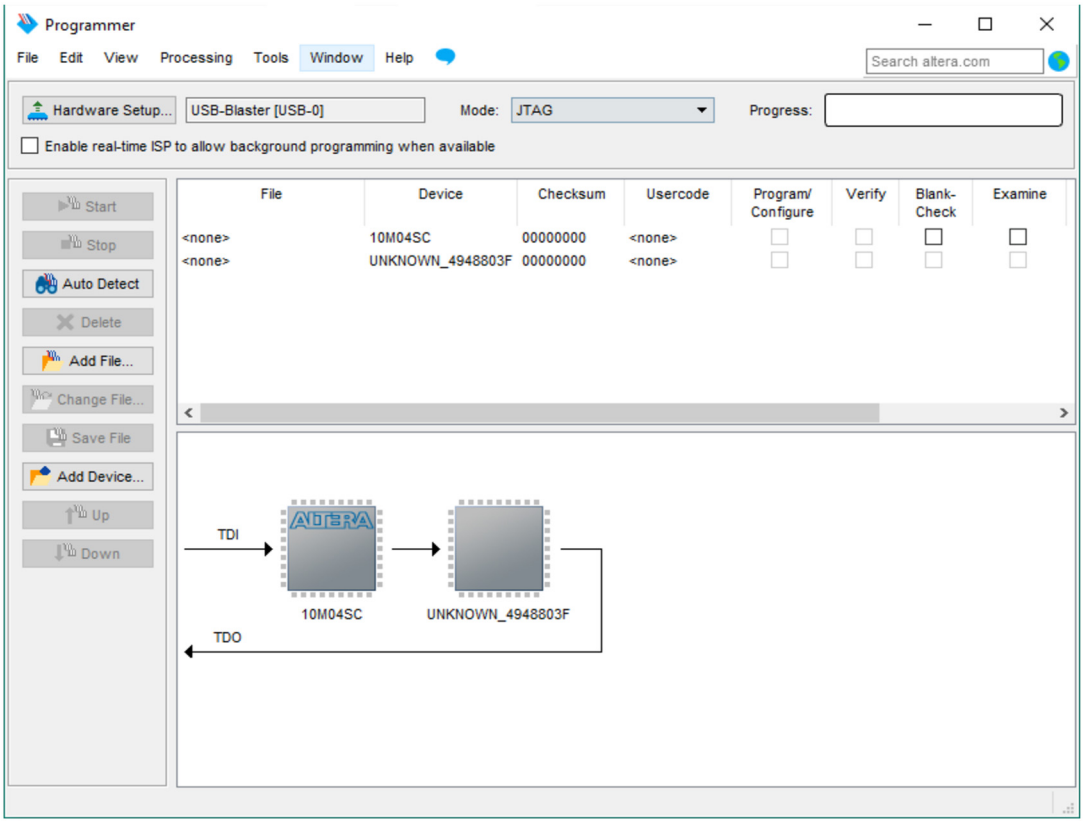


FIGURE A.10

Program window of Altera Quartus, showing that the FPGA and the microcontroller are autodetected.

the program may not run correctly immediately after programming is finished. You need to power the board OFF, and then ON, to make the programmed functions work.

Due to this feature, when you need to program both the microcontroller and the FPGA, always program the microcontroller first, power OFF and ON the board, and then program the FPGA.

A.2.5 CONFIGURING THE VOLTAGE SOURCE

Both then FPGA and the microcontroller need a 3.3V power supply. The HaHa board provides two voltage sources: a fixed 3.3V and an adjustable voltage. The FPGA and the microcontroller have a corresponding slide switch to select the voltage source.

Most often you will need the fixed 3.3V voltage source to obtain the best chip performance. Therefore, please make sure you turn OFF both select switches before you powerup the HaHa board. When you are doing certain experiments that require changing the voltage level, such as the fault injection or

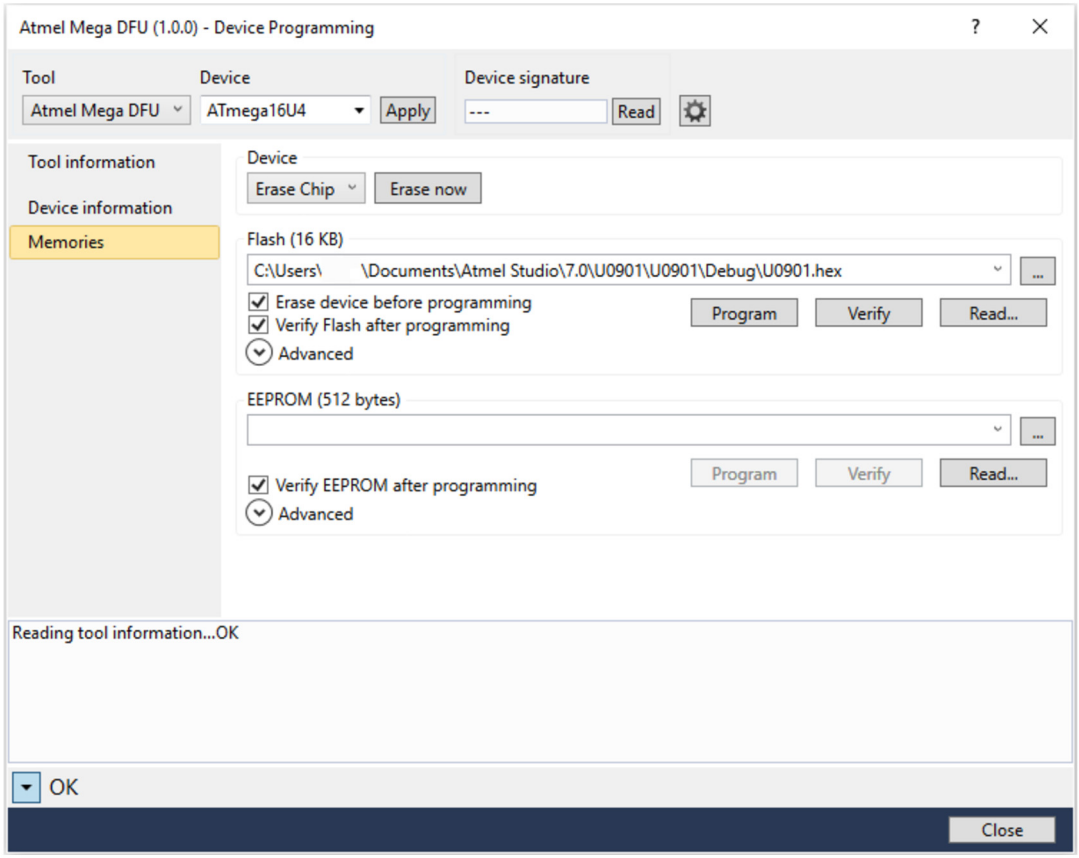


FIGURE A.11

Atmel Studio Device Programming window. It shows that the microcontroller is detected.

the PUF experiment (for reliability assessment of a PUF design), you may need to use the adjustable voltage source. For these experiments, turn on the voltage source selection switch before you powerup the board.

The adjustable voltage has a range from 1.5V to 3.6V. You can control the voltage level by pressing two buttons: SWup and SWdown. There is a total of 64 voltage steps. The regulator can store the voltage value when the power is OFF. It is recommended that every time you finish using the adjustable voltage source, set the voltage back to 3.3V (i.e., the nominal voltage) before you poweroff the board.

A.2.6 CHIP INTERCONNECTIONS

There are, in total, 9 possible interconnections on the board between the two major chips (the FPGA and the microcontroller).

Table A.3 Interconnection between FPGA and microcontroller

Signal Name	Microcontroller Pin No.	FPGA Pin No.	Function
CM0	PD0	29	Data
CM1	PD1	27	Data
CM2	PD2	36	Data
CM3	PD3	25	Data
CM4	PD4	22	Data
CM5	PD5	23	Data
CM6	PD6	21	Data
CM7	PD7	20	Data
CLK_inter	PC7	28	Data or clock

Their pin numbers and functions are listed in Table A.3.

The two chips can communicate through the interconnection fabric to all the devices on the HaHa board. For example, if the microcontroller wants to control the 7-segment display, even though the 7-segment display is not directly connected to it, the FPGA can be configured as an interface connecting the microcontroller and the 7-segment display, so that microcontroller can access it.

A.2.7 USING THE SWITCHES AND LEDs

The HaHa board provides 3 pushbutton switches, 8 slide switches (sliders), and 8 LEDs, all connecting to the FPGA. There is only one LED connected to the microcontroller.

For the 3 pushbutton switches connected to the FPGA, each one provides a low logic level (0 volts) when it is not pressed, and provides a high logic level (3.3 volts) when pressed. Table A.4 lists the corresponding pin numbers on the FPGA.

Table A.4 Pushbutton switch pin number list

Signal Name	FPGA Pin No.	Description
SW1	93	Pushbutton [1]
SW2	96	Pushbutton [2]
SW3	97	Pushbutton [3]

The 8 slide switches on the HaHa board 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 ON, it provides a low logic level (0 volts) to the FPGA, and when it is OFF, it provides a high logic level (3.3 volts). Table A.5 lists the pin numbers connected to the switches.

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 high logic level turns the LED on, and driving the pin to low logic level turns the LED off. Table A.6 lists the pin numbers connected to the LEDs.

Table A.5 Slide switches pin number list

Signal Name	FPGA Pin No.	Description
S0	44	Slide Switch [0]
S1	43	Slide Switch [1]
S2	42	Slide Switch [2]
S3	41	Slide Switch [3]
S4	40	Slide Switch [4]
S5	39	Slide Switch [5]
S6	38	Slide Switch [6]
S7	33	Slide Switch [7]
S8	32	Slide Switch [8]
S9	30	Slide Switch [9]

Table A.6 User LED pin number list

Signal Name	FPGA Pin No.	Description
D1	141	LED [1]
D2	140	LED [2]
D3	135	LED [3]
D4	133	LED [4]
D5	132	LED [5]
D6	131	LED [6]
D7	129	LED [7]
D8	127	LED [8]

A.2.8 USING THE 7-SEGMENT DISPLAY

The HaHa board includes one 7-segment display. As indicated in Fig. A.12, the seven segments are connected to different pins on the FPGA. Applying a low logic level to a segment causes it to light up. On the contrary, applying a high logic level turns it off.

Each segment in the display is identified by an index from **a** to **g**, with the corresponding position shown in Fig. A.12. In addition, the decimal point is identified as DP. Table A.7 shows the connections between the FPGA pins to the 7-segment display.

A.2.9 USING THE EXPANSION HEADERS

The HaHa board provides 3 expansion headers. Two of them have 40 pins, and one has 20 pins. There is a total of 52 FPGA input-output (I/O) pins connected to the header pins.

A.2.10 CLOCK CIRCUITRY

The HaHa board includes a 50 MHz clock signal for the FPGA, and an 8 MHz clock signal for the microcontroller.

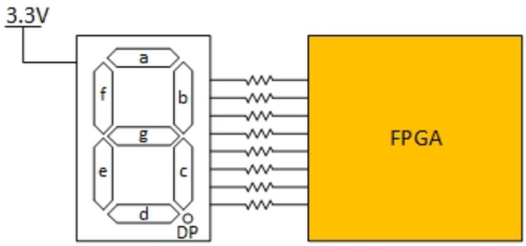


FIGURE A.12

Connections between the 7-segment display and the FPGA.

Table A.7 Pin assignments for the 7-segment display

Display Index	Signal Name	FPGA Pin No.
a	D11	123
b	D12	120
c	D13	119
d	D14	118
e	D15	117
f	D16	116
g	D17	111
DP	D18	110

The clock connected to the FPGA is used for clocking the user logic. In addition, the clock input is connected to the phase-locked 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 Section A.2.6, there is a chip interconnection that can also be used as the clock source for the FPGA. The I/O pin will output clock signal for the FPGA when the user configures the microcontroller. The reconfiguration will enable the two chips to be synchronized to the same clock.

Various I/O pins on the FPGA can be used as the clock input pin. I/O pins connected to the expansion header can connect to external clock signals, if needed.

Pins that can be used as external clock inputs are listed in Table A.8.

A.2.11 USING SPI DEVICES

There are three SPI devices that can work as an SPI slave, controlled by the microcontroller: an EEPROM, an accelerometer, and a Bluetooth processor. There are also two SPI slots on the HaHa board. Users can connect more devices to the microcontroller through the slots.

Note: SPI pin names–MISO, MOSI, SCK and SS′ are the names used by AVRs. Other devices may use a different set of names. Check the datasheet of the particular device you are using to learn the SPI pin names.

The SPI setup: the microcontroller has three simultaneous slaves. Users can select which slave to communicate by asserting the SS′ (Slave Select) pin of the slave devices. The I/O pins used for slave

Table A.8 FPGA clock input pins and their connections

FPGA Clock Input Pin	FPGA Pin No.	Signal Name	Function Description
CLK0n	25	CM3	Chip interconnection
CLK0p	26	CM2	Chip interconnection
CLK1n	27	CM1	Chip interconnection
CLK1p	28	CLK_inter	Chip interconnection
CLK6n	51	H_A_2	Expansion header pin
CLK6p	52	H_B_3	Expansion header pin
CLK7n	53	H_A_3	Expansion header pin
CLK7p	55	H_B_4	Expansion header pin
CLK2n	88	CLK_50M	50 MHz clock source
CLK2p	89	H_B_17	Expansion header pin
CLK3n	90	H_2B_17	Expansion header pin
CLK3p	91	H_2B_18	Expansion header pin

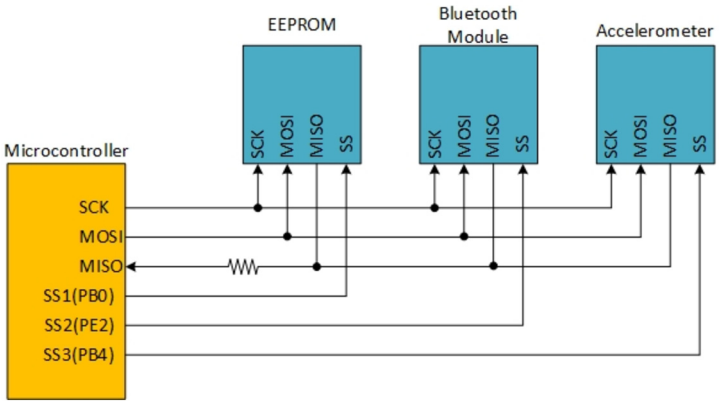


FIGURE A.13

Setup used with SPI interface on the HaHa board.

selection is shown in Fig. A.13. When selecting one of the slaves to communicate, assert the SS′ signal of that slave device by setting it Low. In order to avoid conflict, SS′ for other devices must be set to High. Note that the microcontroller can only communicate with one SPI device at a time.

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

Figure A.14 shows the SPI slot pin configurations. A user can incorporate additional SPI devices by connecting corresponding pins to the slot. He/she can also use it as locations to probe signals on the board.

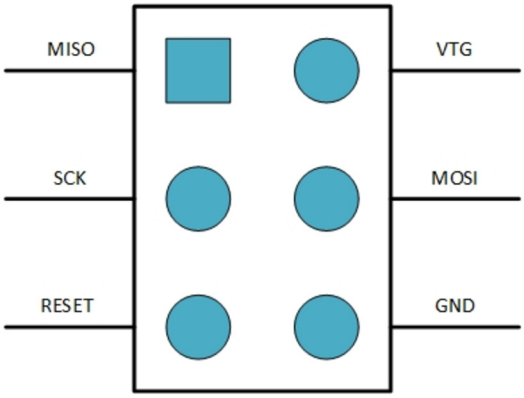


FIGURE A.14

SPI header pin location.

A.2.12 SIDE-CHANNEL MEASUREMENT

There is a dedicated area as shown in Fig. A.15A designed for side-channel (example, power or current) measurement on the board. A precise 1 Ohm current sensing resistor is mounted between the FPGA and the power supply source. By measuring the voltage drop across the resistor, the current which represents the power consumption can be obtained. SubMiniature version A (SMA) connectors are also provided.

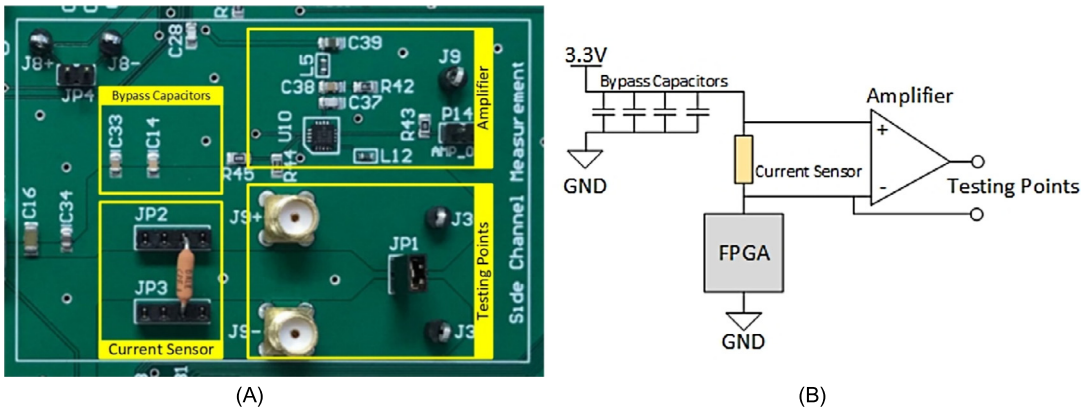


FIGURE A.15

(A) Side-channel measurement area on the HaHa board; (B) The schematic diagram showing two measurement options: with or without amplification.

The side-channel measurement circuit integrates an instrumentation amplifier. It can amplify the voltage difference across the current-sensing resistor by four times. The bandwidth of the part is as high as 2.4 GHz. The voltage drop can be measured either across the sense resistor or across the amplifier. The latter provides an increased SNR, as shown in Fig. A.15B.

A.3 EXAMPLES FOR PROGRAMMING THE FPGA AND THE MICROCONTROLLER

This section provides a number of example circuits and codes, which can be mapped into the FPGA and microcontroller chips, respectively. These circuits/codes are intended to serve as practice runs before the actual experiments are performed. The examples are designed to demonstrate some of the basic features of the board, such as, operation of the output LEDs and input switches. The source codes (Verilog and Assembly code) are provided for each demonstration.

A.3.1 PROGRAMMING THE FPGA

A.3.1.1 A Counter Implemented in the FPGA

Implementing a simple circuit into the FPGA: the module maps a 4-bit up counter into the FPGA, and connects it to 3 inputs and 1 output. One input works as the clock, one resets the output to 0000, and the other input enables counting. The output is a 4-bit binary number representing the value counted. Therefore, 1 clock source, 2 switches, and 4 LEDs will be used. The steps are as follows:

1. Create a new project in Quartus. Select the right device that the HaHa board uses, which is 10M50SAE144C8G, as shown in Fig. A.16.

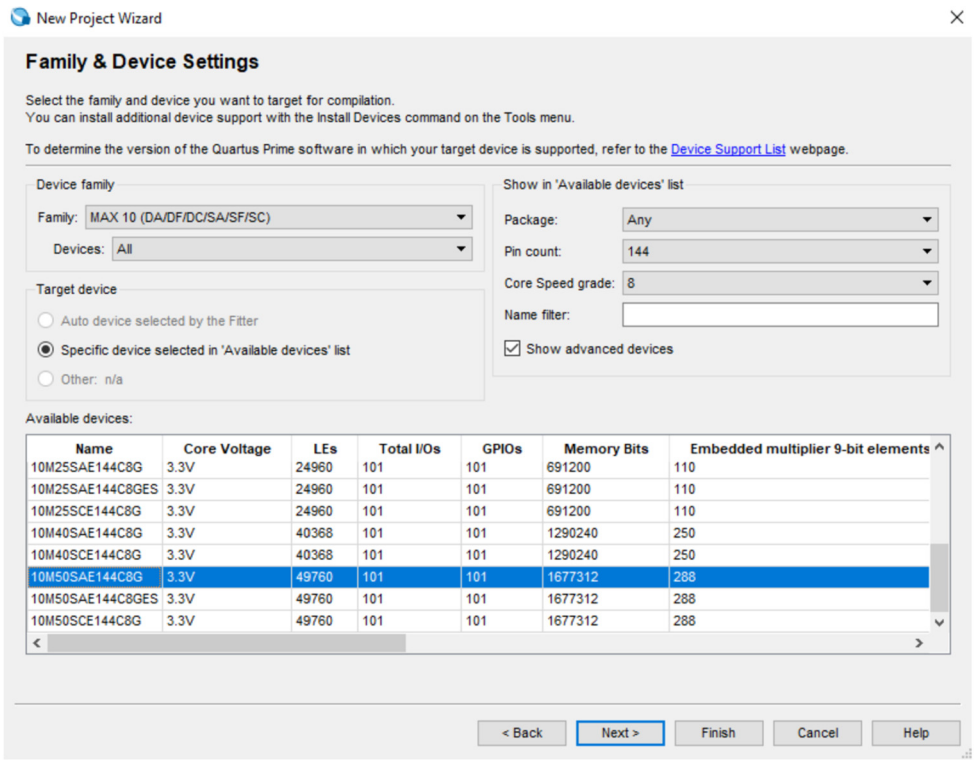


FIGURE A.16

Selecting the device on the HaHa board.

2. Create a new Verilog hardware description language (HDL) file and add it to the project. Create the up counter module in the v-file. The code is given below. The code contains two modules: the up counter module and a sub-module, which is a frequency divider. This example uses a 50 MHz clock source, which is considerably faster than a human eye can observe the counted values. Hence, the frequency divider makes the frequency of count operation much slower. Type in the code, then perform analysis and elaboration of the code.

Verilog Code

```
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_slow) 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 the frequency divider submodule that makes the clock frequency slower, so that LED output can be
observed 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'b00111110101111000010000000)
            begin
                clkout=~clkout;
                cnt=26'b0;
            end
    end
endmodule
```

3. Assign signals to the right pins. Open the pin planner window as shown in Fig. A.17. Assign the signals to the right location according to the HaHa user manual.

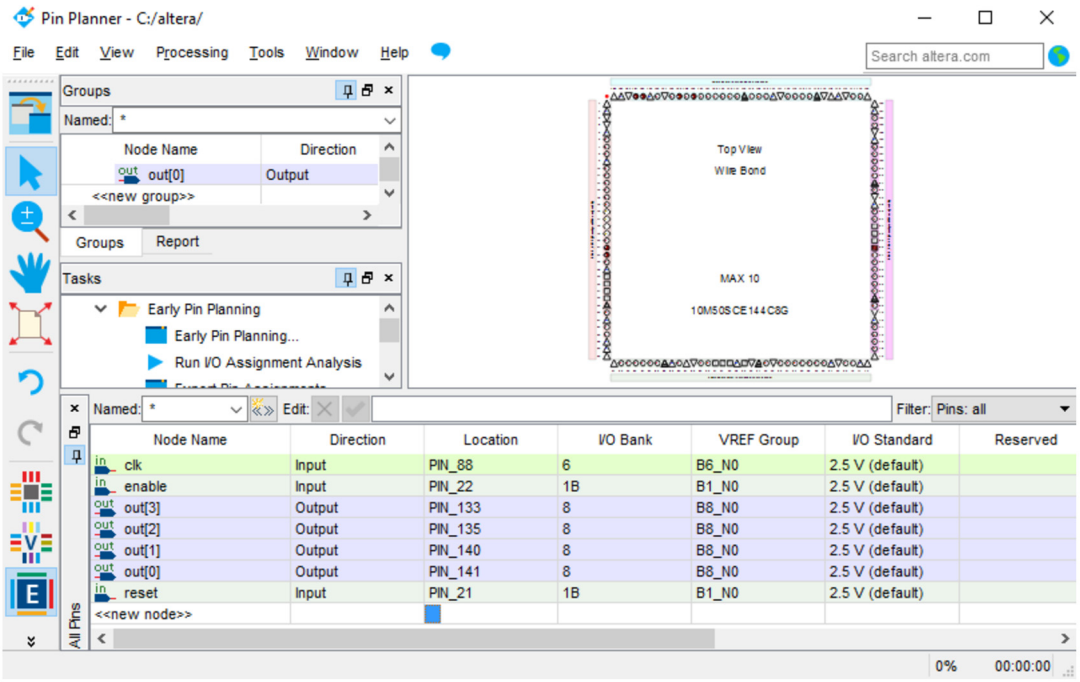


FIGURE A.17

Pin planner window in Quartus.

4. Start compilation. When finished, the SOF-file will be created. This file is a representation of the FPGA bitstream produced by the vendor tool.
5. Open the programmer window. Use the SOF-file to program the FPGA.

A.3.1.2 Arithmetic and Logic Unit

This example shows mapping of a simple arithmetic-logic unit (ALU) into the FPGA. The ALU has two 4-bit inputs **a** and **b**, and one 2-bit function selection input **sel**. The ALU performs various operations according to different input values at the function selection input.

Selection bits	Function (o)
00	a plus b
01	a multiply b (only the least 4 bits)
10	a AND b (bitwise)
11	an OR b (bitwise)

Input ports can be used to provide the operands and selection bits. Switches and buttons on the HaHa board can be used for this purpose. The outputs can be displayed on four LEDs.

Using Verilog HDL, the ALU can be described in several ways, such as using ‘case’ statement, as shown in the example code below.

Verilog Code

```
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;
            2'b10: o = a & b;
            2'b11: o = a | b;
            default: o = a + b;
        endcase
    end

endmodule
```

After compilation, use pin planner to assign the I/O ports to suitable pins of the FPGA, as shown in Fig. A.18.

Node Name	Direction	Location
a[3]	Input	PIN_21
a[2]	Input	PIN_22
a[1]	Input	PIN_23
a[0]	Input	PIN_25
b[3]	Input	PIN_26
b[2]	Input	PIN_27
b[1]	Input	PIN_29
b[0]	Input	PIN_30
o[3]	Output	PIN_141
o[2]	Output	PIN_140
o[1]	Output	PIN_135
o[0]	Output	PIN_133
sel[1]	Input	PIN_85
sel[0]	Input	PIN_86

FIGURE A.18

ALU pin plan.

Verify that the function is implemented correctly by generating the FPGA bitstream file and programming the FPGA.

A.3.1.3 Finite State Machine

A finite-state machine (FSM) or simply a state machine is a mathematical model of computation used to realize sequential logic circuits. It represents an abstract machine that can perform transition through a finite number of states, and is typically used to implement control circuits in a design.

Example: One of the University of Florida parking lots has only one entrance gate and it 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 exiting. A sign to indicate if the parking lot is FULL or FREE is positioned at the entrance.

Verilog Code

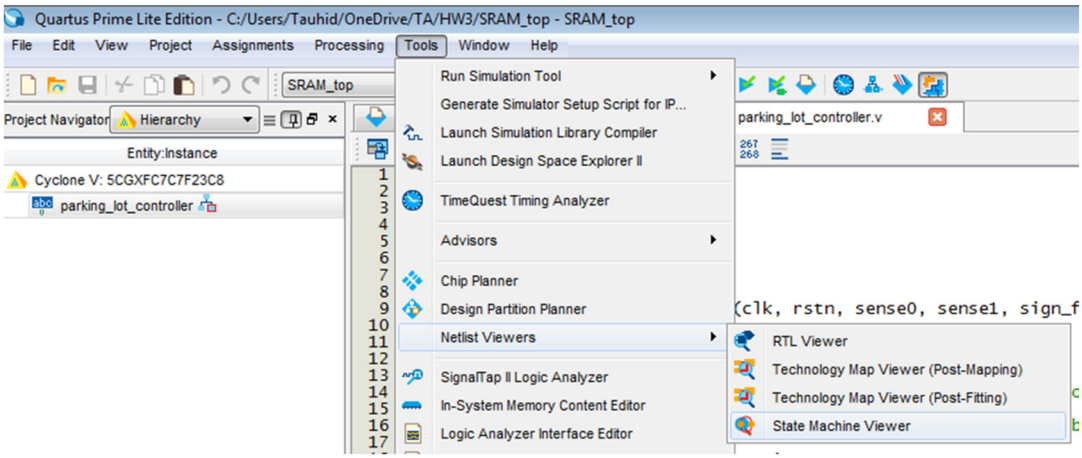
```
module parking_lot_controller (clk, rstn, sense0, sense1, sign_full, sign_free);

    // 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)
    // 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
            state = 4'b0001;
        end
        // car has crossed both sensors
        else if (sense0 == 1'b0 && sense1 == 1'b0) begin
            state = 4'b0010;
        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'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
                        if (sense0 == 1'b0 && sense1 == 1'b0) begin
                            state = 4'b1000;
                        end
                        // car has finished crossing inside sensor, and is passing outside sensor
                        else if (sense0 == 1'b0 && sense1 == 1'b1) begin
                            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 count and go to initial state
                        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
                        state = 4'b0000;
                    end
                end
            end
endmodule
```

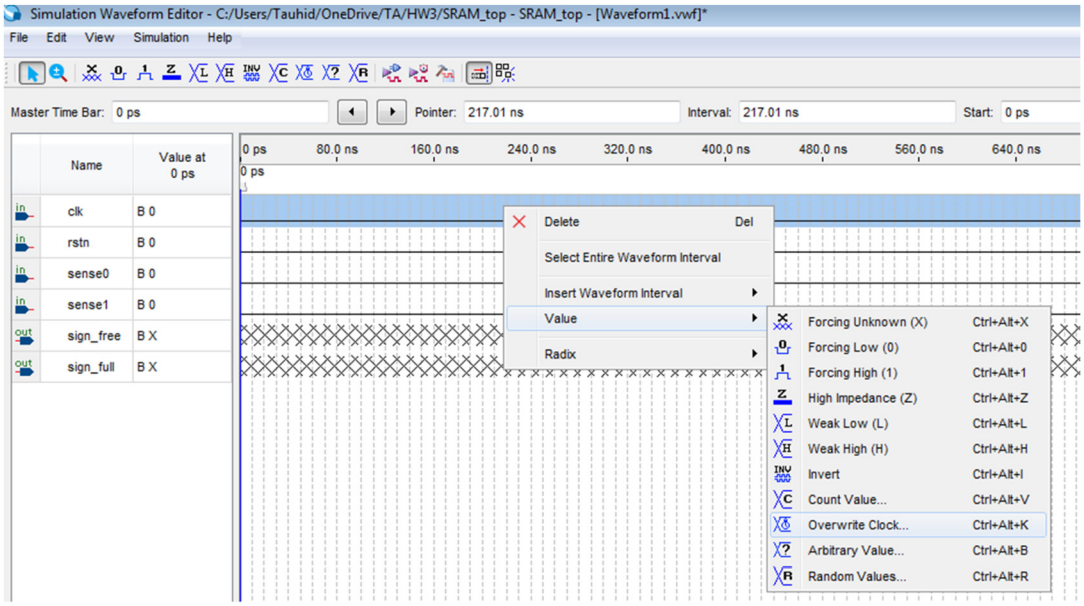
State Diagram:

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



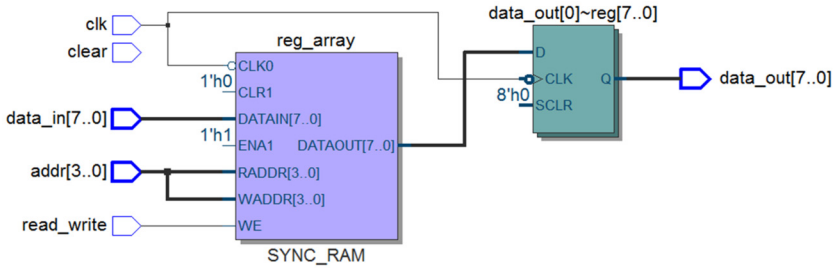
Simulation:

Go to New → University Program VWF → Edit → Insert → Insert Node or Bus → Node Finder → list → Select all → Insert values for the inputs → Simulation.



A.3.1.4 Read-Write Operation of an SRAM

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



Verilog Code for SRAM read-write operation

```
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!
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;
      data_out = reg_array[addr];
end
endmodule
```

A.3.2 PROGRAMMING THE MICROCONTROLLER

A.3.2.1 Counter Programmed in the Microcontroller

This tutorial programs a simple counter into the microcontroller. The function is similar to that of the circuit described in Section A.3.1, except that this time it is carried out in relation to the microcontroller. Since the microcontroller is not connected to many switches, and LEDs, due to its I/O pin limitation, it gets access to the peripherals through the FPGA, which works as an interface between them. The steps are described below:

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

Assembly Code

```
    .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

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 the values clearly
    rjmp loop
;the code 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. A HEX-file will be created once the project is built.
4. Program the microcontroller with the HEX-file as shown in Fig. A.19.

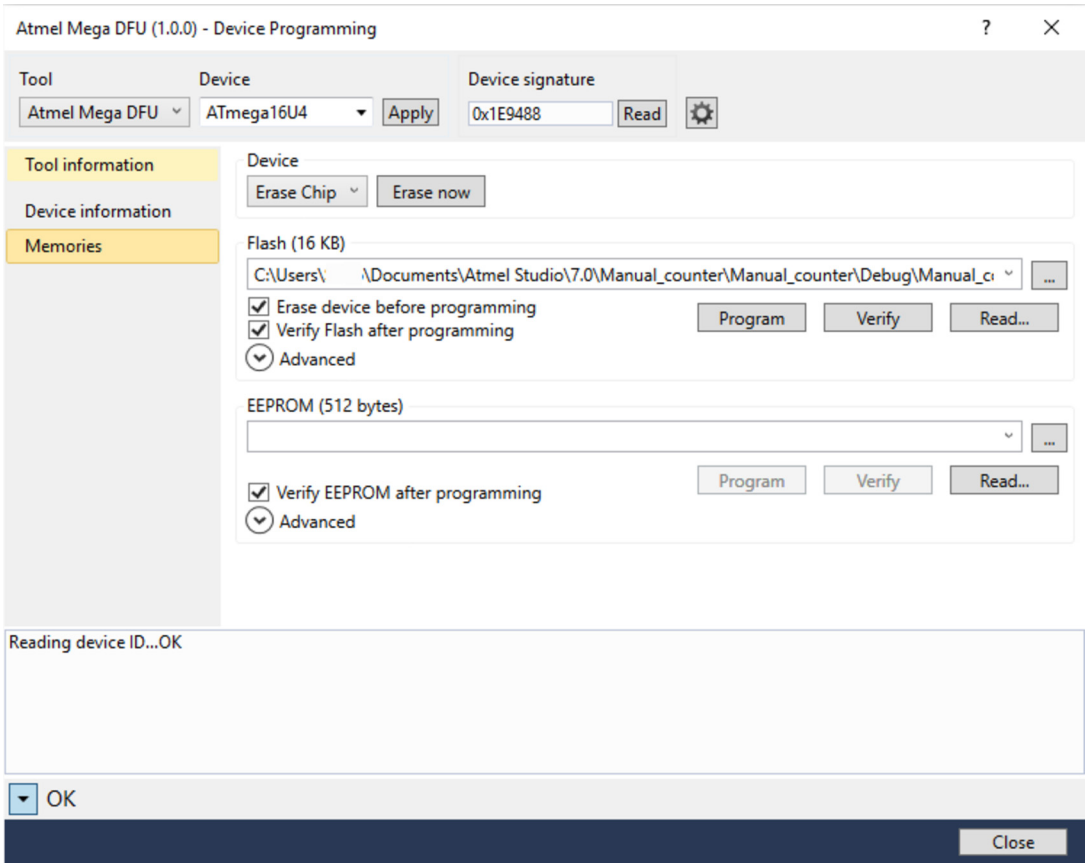


FIGURE A.19

Microcontroller programming window.

5. Power the HaHa board OFF and On. Program the FPGA so that 4 LEDs are directly connected to the microcontroller's I/O pins PD0 to PD3. Two switches are directly connected to the microcontroller's I/O pins PD4 and PD5. This will require repeating the steps in Section A.3.1, but with different code.

A.3.2.2 Light Up a LED Through an I/O Port

The user has to configure the pin first before using an I/O port. 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.

In the HaHa board, there is a red LED connected to PORTB7. Therefore, configure this port as output using code:

```
sbi DDRB, 7
```

Make the 7th bit of PORTB to be "1" to produce a high level on that port:

```
sbi PORTB, 7
```

Finish the program by entering into an endless loop:

```
Loop: rjmp loop
```

The program code is shown below.

Assembly Code

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

main:
    sbi DDRB, 7
    sbi PORTB, 7

loop:
    rjmp loop
```

A.3.2.3 Make the LED 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:

```
cbi PORTB, 7
```

One cannot see the LED blink if the LED blinks at the frequency of the chip's internal clock. To slow it down, a delay can be inserted.

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

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

In the beginning of the delay function, the value of the working register r23 is initialized to 0x00 by using ldi instruction. Then, its value 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. When the bit value is 1, the following instructions will be skipped. The code of the whole program is given below.

Note: The delay time provided in this program is still not long enough for human eyes to see it clearly. Users can make the delay longer by calling another delay in the delay loop.

Assembly Code

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

main:
    ldi r16, 0xFF
    out DDRB, r16


loop:
    sbi PORTB, 7
    call delay
    cbi PORTB, 7
    call delay
    rjmp loop

delay:
    ldi r17, 0x00
delay_inc:
    inc r17
    sbrs r17, 7
    rjmp delay_inc
    ret
```

A.3.2.4 Read a Value From an SRAM Address and Send It to I/O Ports

There is an embedded SRAM array inside the microcontroller. Users can read a value from an SRAM address and store the value to a working register by inserting the following code:

```
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
```

The complete program is as follows:

Assembly Code

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

main:
    ldi r16, 0xFF
    out DDRD, r16

    lds r17, 300
    out PORTD, r17
loop:
    rjmp loop
```

A.3.2.5 Program Controlled by an Input Port

Occasionally, a program might need to read signals from an I/O port, which is configured as an input port. For example, let us consider a scenario, where a program is waiting for a high level, that is, “1” from an input port to start the execution. If the signal coming into the port is always low, “0”, then the program will not be executed.

Assembly Code

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

main:
    cbi, DDRC, 7
```

```
loop:
    sbis PINC, 7
    rjmp loop

    ; below is the code, which will be executed when PC7=1
```

A.3.2.6 IO Value to Control an LED

An user can propagate a value from an input port to an output port. For example, let us consider an input port connected to a switch, and an output port connected to a LED. In this case, the switch can control the LED by propagating 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
```

A.3.2.7 Simple Programming Example

The instruction set of the microcontroller consists of arithmetic and logic instructions. Arithmetic instructions include ADD, SUB, and MUL. This example will show how to implement a simple function using several working registers. Let us consider a function: f(x) = 255 - 2x, if x < 128; otherwise f(x) = 0.

The required Instructions for the example include MUL, SUB, and a branch. The result will be visible as the output of portD. The 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
```

A.4 DESIGN SPECIFICATIONS

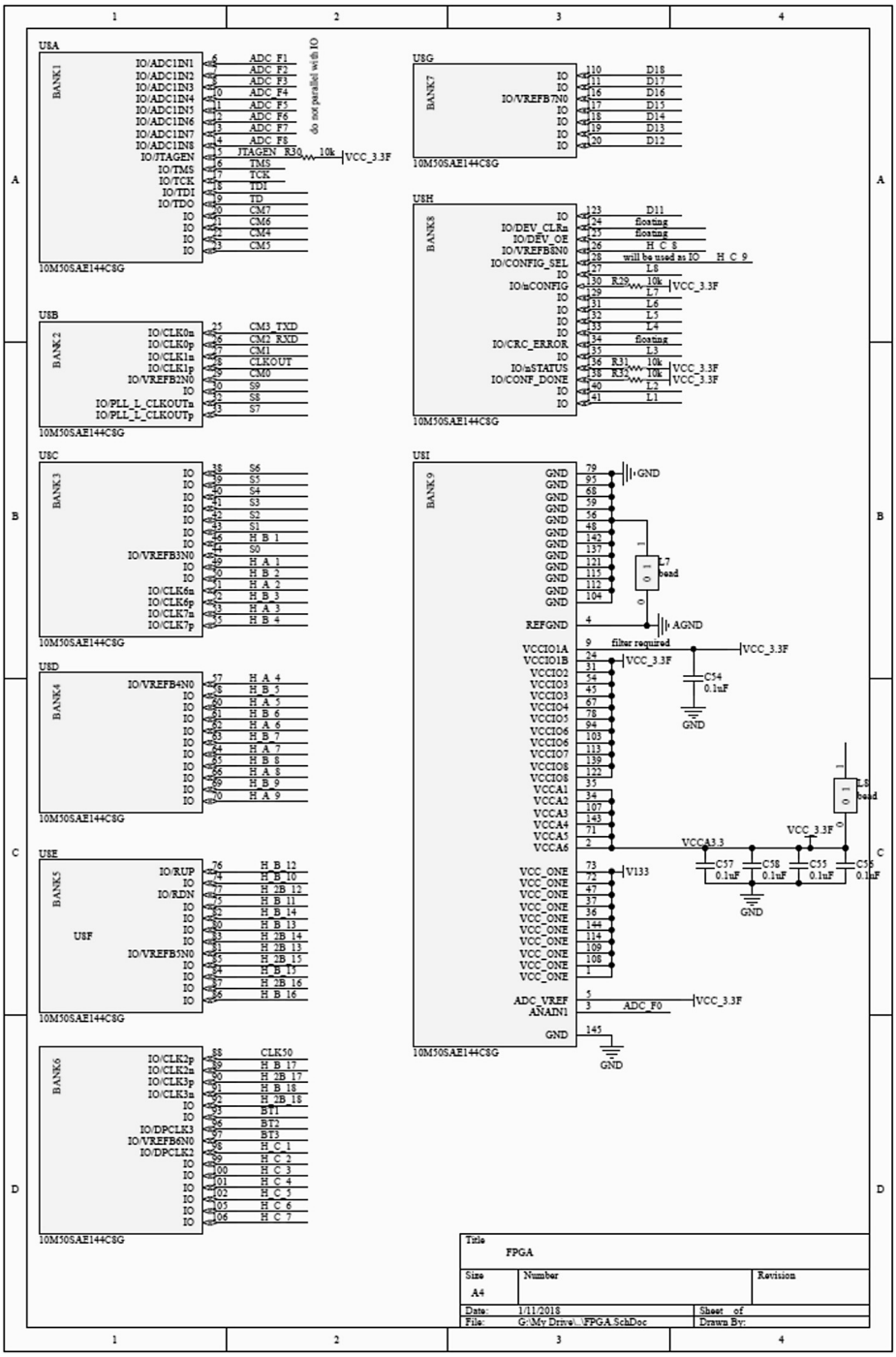
This section provides the bill-of-materials (Table A.9), schematic, assembly diagram (Fig. A.20) and layout (Fig. A.21) of the HaHa board (version 2).

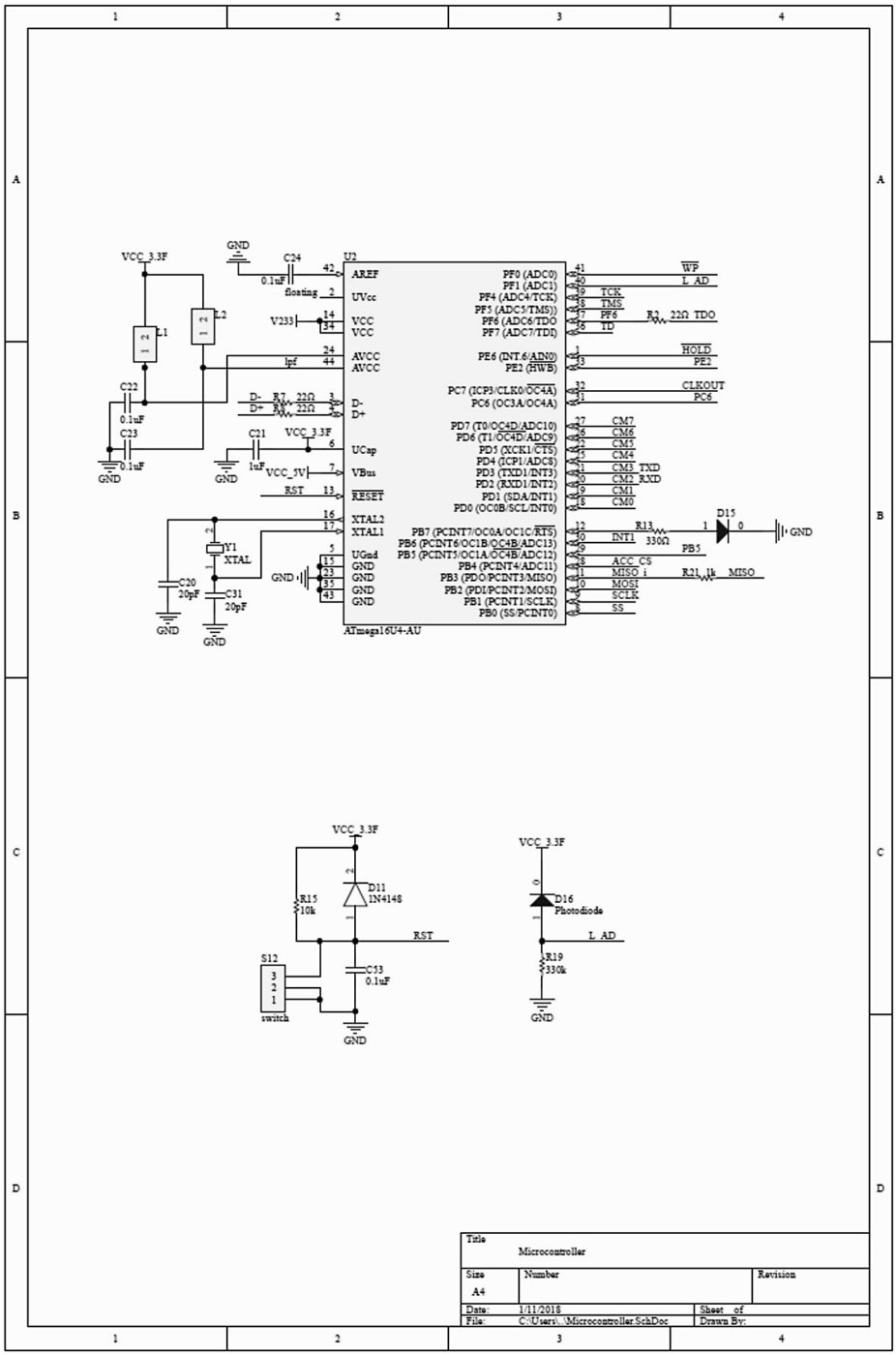
Table A.9 Bill-of-materials (BOM)

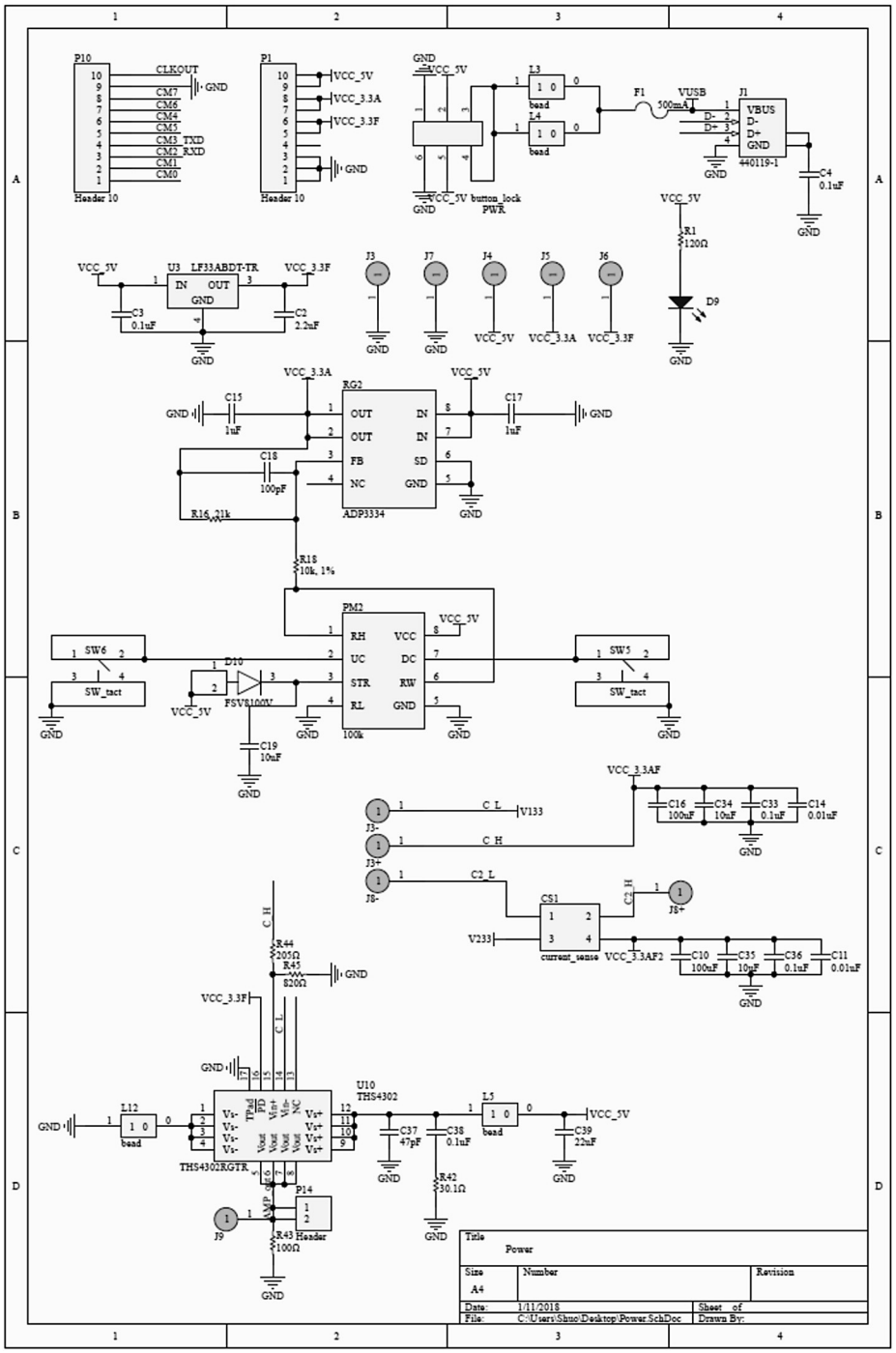
Manufacturer	Manufacturer No	Quantity	Designator
Omron Electronics	XM2C-0942-112L	1	J10
CUI	UJ2-BH-1-TH	2	J1, J18
Texas Instruments	THS4302RGTR	1	U10
Microchip	RN4870-V/RM118	1	U7
C&K Components	PTS645SM43SMTR92 LFS	6	SW1, SW2, SW3, SW4, SW5, SW6
C&K Components	OS102011MS2QN1	15	S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14
Optek / TT Electronics	OP980	1	D16
TDK Corporation	MPZ1608S101ATAH0	6	L3, L4, L5, L7, L8, L12
Apem	MHPS2283	1	PWR
Microchip	MCP2200-I/SS	1	U4
LVK Series	LVK12R010DER	1	CS1
Lite on	LTST-C194KSKT	1	D14
Lite on	LTST-C190KRKT	6	D5, D6, D7, D8, D9, D12
Lite on	LTST-C190KGKT	5	D1, D2, D3, D4, D13
Lite on	LTL2R3KRD-EM	1	D15
Lite on	LSHD-7501	1	DS1
STMicroelectronics	LIS2DE12TR	1	U5
STMicroelectronics	LF33ABDT-TR	1	U3
Taiyo Yuden	LBC3225T100KR	5	L1, L2, L9, L10, L11
Fairchild Semiconductor	FSV8100V	1	D10
Fox	FOXSDLF/080-20	1	Y1
Panasonic	ERJ-P06D8200V	1	R45
Panasonic	ERJ-6GEYJ560V	5	R17, R37, R38, R39, R40
Panasonic	ERJ-6GEYJ472V	2	R27, R41
Panasonic	ERJ-6GEYJ471V	1	R25
Panasonic	ERJ-6GEYJ470V	5	R24, R33, R34, R35, R36
Panasonic	ERJ-6GEYJ223V	3	R9, R10, R14
Panasonic	ERJ-6GEYJ220V	3	R2, R7, R8
Panasonic	ERJ-6GEYJ103V	9	R3, R4, R6, R15, R22, R29, R30, R31, R32
Panasonic	ERJ-6GEYJ102V	2	R21, R26
Panasonic	ERJ-6GEYJ101V	1	R43
Panasonic	ERJ-6ENF3303V	1	R19
Panasonic	ERJ-6ENF3302V	3	R11, R12, R23
Panasonic	ERJ-6ENF3300V	2	R13, R28
Panasonic	ERJ-6ENF2102V	1	R16
Panasonic	ERJ-6ENF2050V	1	R44

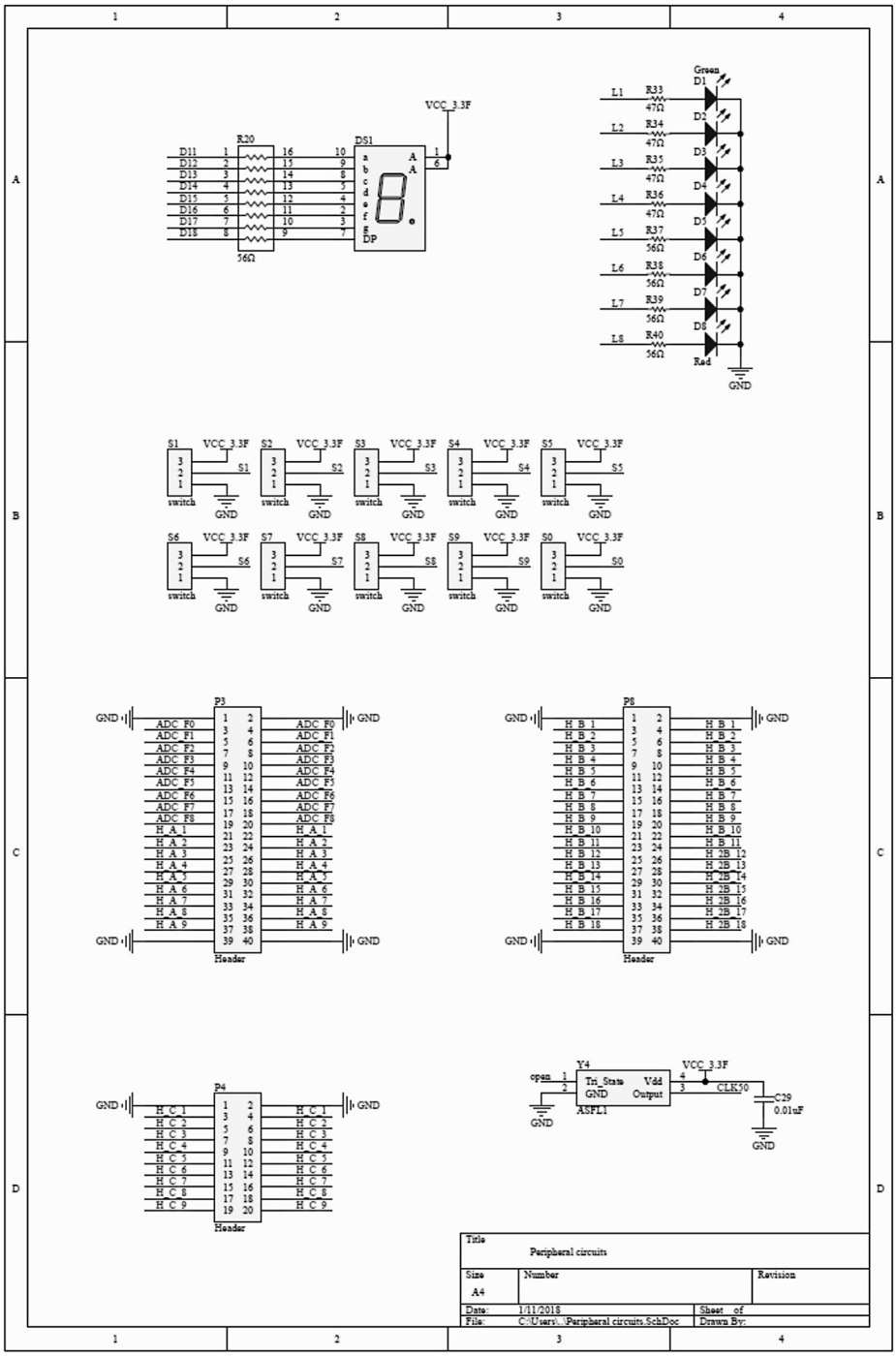
Table A.9 (*continued*)

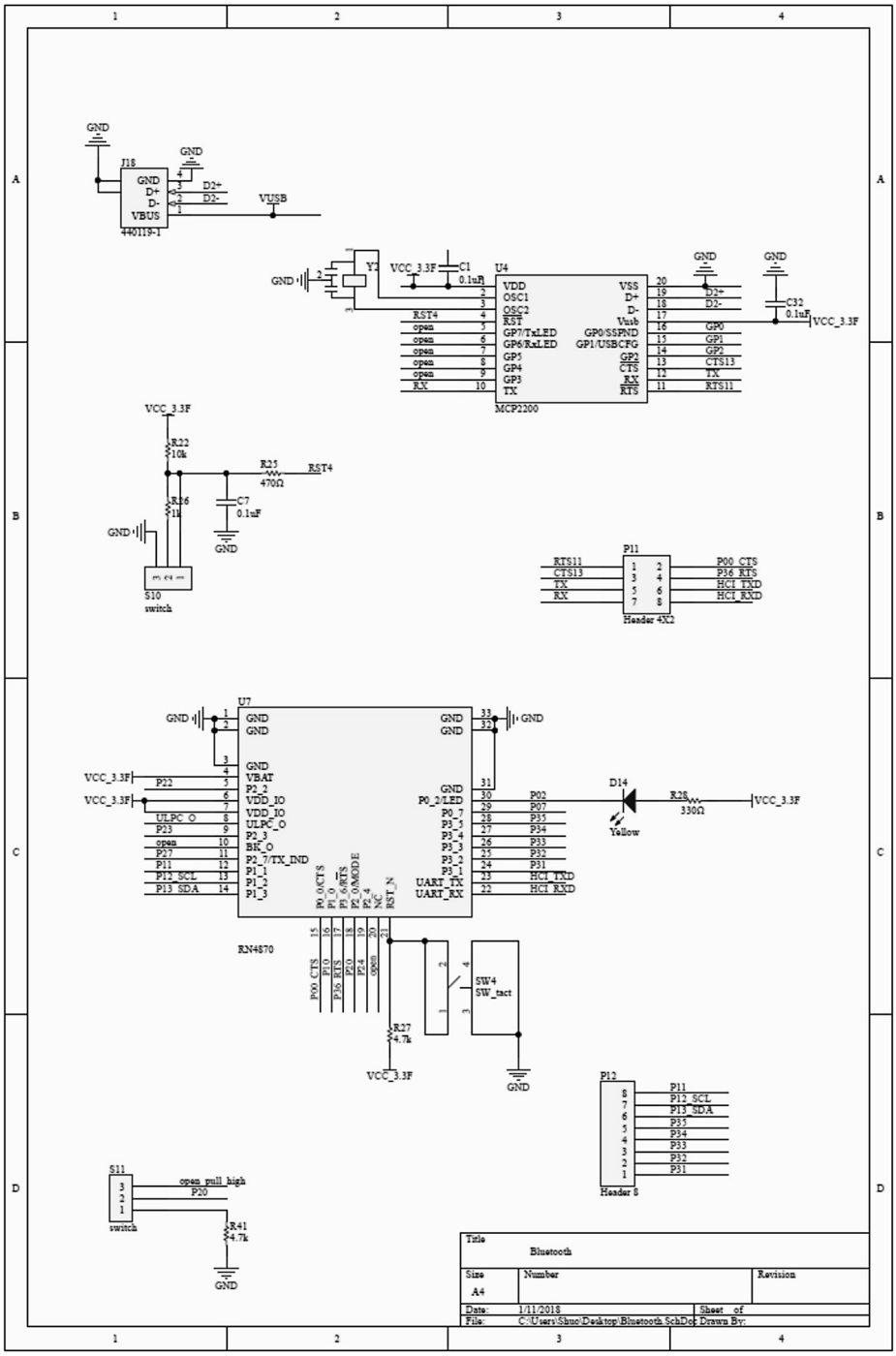
Manufacturer	Manufacturer No	Quantity	Designator
Panasonic	ERJ-6ENF1200V	1	R1
Panasonic	ERJ-6ENF1002V	1	R18
Panasonic	ERJ-6ENF68R0V	1	R5
Panasonic	ERJ-6ENF30R1V	1	R42
Maxim Integrated	DS1809U-100+	1	PM2
Murata Electronics	CSTCE12M0G55-R0	1	Y2
Kemet	C1206C107M9PACTU	2	C10, C16
Kemet	C0805C470J5GACTU	1	C37
Kemet	C0805C226M9PACTU	1	C39
Kemet	C0805C225K8RACTU	1	C2
Kemet	C0805C200J1GACTU	2	C20, C31
Kemet	C0805C106K8PACTU	4	C19, C26, C34, C35
Kemet	C0805C105K4RACTU	6	C5, C6, C8, C15, C17, C21
Kemet	C0805C104J5RACTU	23	C1, C3, C4, C7, C9, C12, C13, C22, C23, C24, C25, C27, C30, C32, C33, C36, C38, C53, C54, C55, C56, C57, C58
Kemet	C0805C103K5RACTU	3	C11, C14, C29
Kemet	C0805C101J3GACTU	2	C18, C28
Microchip Technology / Atmel	ATmega16U4-au	1	U2
Abracon LLC	ASFL1-50.000MHZ-EC-T	1	Y4
Analog Devices	ADP3334ARMZ-REEL7	1	RG2
Analog Devices	ADM3202ARUZ-REEL7	1	U9
Molex	702471051	1	P5
3M	961240-6404-AR	2	P3, P8
3M	961220-6404-AR	1	P4
3M	961208-6404-AR	1	P11
3M	961206-6404-AR	3	P2, P9, P13
3M	961110-6404-AR	2	P1, P10
3M	961108-6404-AR	1	P12
3M	961102-6404-AR	3	JP1, JP4, P14
3M	929870-01-04-RA	2	JP2, JP3
Keystone Electronics	5006	18	J2, J3, J3+, J3-, J4, J5, J6, J7, J8+, J8-, J9, J11, J12, J13, J14, J15, J16, J17
Bourns	4816P-1-560LF	1	R20
Bel Fuse	0697H1000-02	1	F1
Nexperia	74HC3G14DC-Q100H	1	U1
Microchip	25LC1024-E/SM	1	U6
Intel / Altera	10M50SAE144C8G	1	U8
TE Connectivity	5-1814832-1	2	J9+, J9-
Fairchild Semiconductor	1N4148	1	D11

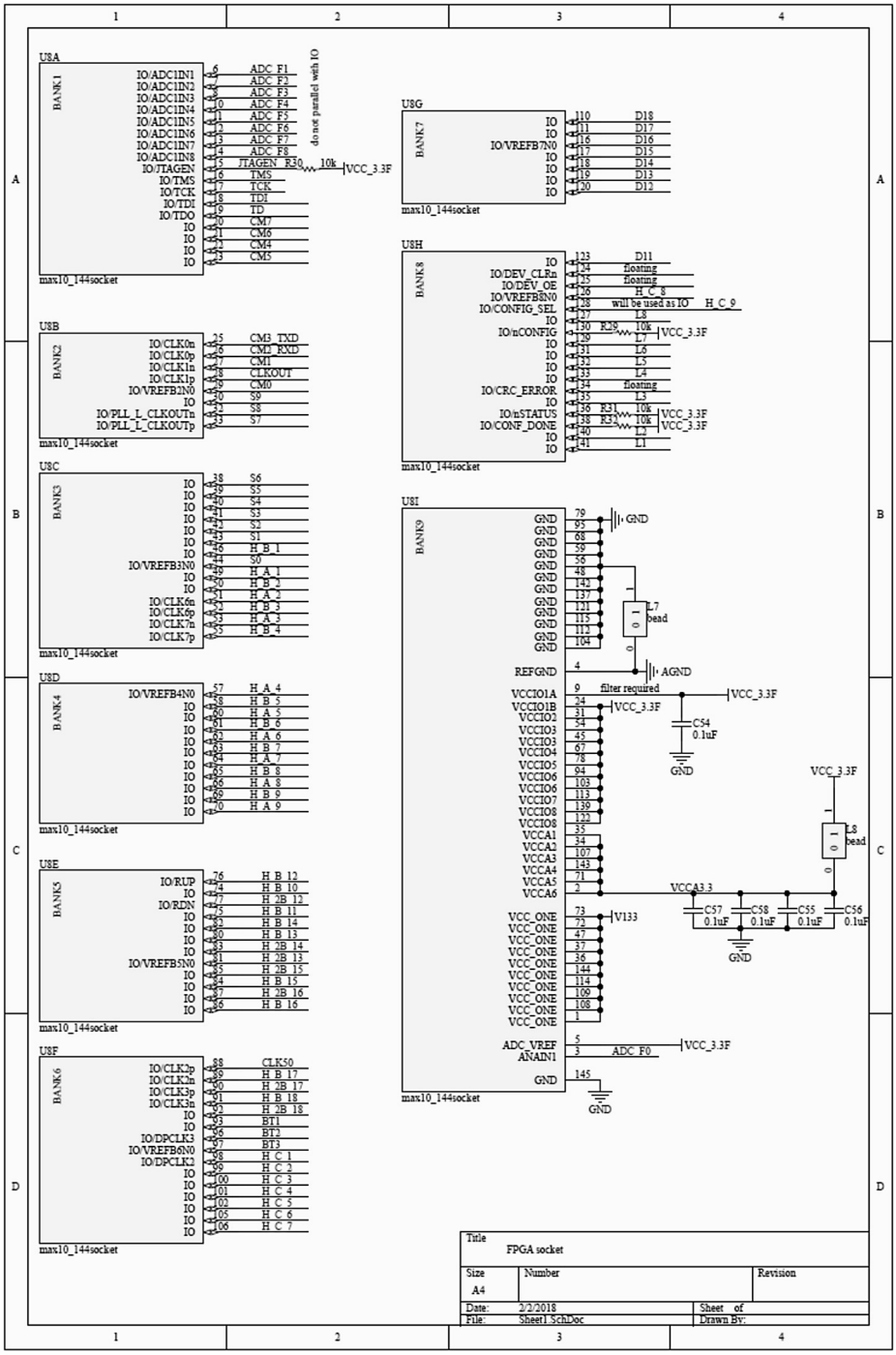

U8A
BANK1
IO/ADC1IN1
IO/ADC1IN2
IO/ADC1IN3
IO/ADC1IN4
IO/ADC1IN5
IO/ADC1IN6
IO/ADC1IN7
IO/ADC1IN8
IO/JTAGEN
IO/TMS
IO/TCK
IO/TDI
IO/TDO
IO
IO
IO
IO
ADC_F1
ADC_F2
ADC_F3
ADC_F4
ADC_F5
ADC_F6
ADC_F7
ADC_F8
do not parallel with IO
JTAGEN_R30
10k
VCC_3.3F
TMS
TCK
TDI
TD
CM7
CM6
CM4
CM5
10M50SAE144C8G
U8B
BANK2
IO/CLK0n
IO/CLK0p
IO/CLK1n
IO/CLK1p
IO/VREFB2N0
IO
IO/PLL_L_CLKOUTn
IO/PLL_L_CLKOUTp
CM3_TXD
CM2_RXD
CM1
CLKOUT
CM0
S9
S8
S7
10M50SAE144C8G
U8C
BANK3
IO/VREFB3N0
IO/CLK6n
IO/CLK6p
IO/CLK7n
IO/CLK7p
S6
S5
S4
S3
S2
S1
H_B_1
S0
H_A_1
H_B_2
H_A_2
H_B_3
H_A_3
H_B_4
10M50SAE144C8G
U8D
BANK4
IO/VREFB4N0
H_A_4
H_B_5
H_A_5
H_B_6
H_A_6
H_B_7
H_A_7
H_B_8
H_A_8
H_B_9
H_A_9
10M50SAE144C8G
U8E
BANK5
U8F
IO/RUP
IO/RDN
IO/VREFB5N0
H_B_12
H_B_10
H_2B_12
H_B_11
H_B_14
H_B_13
H_2B_14
H_2B_13
H_2B_15
H_B_15
H_2B_16
H_B_16
10M50SAE144C8G
U8G
BANK7
IO/VREFB7N0
D18
D17
D16
D15
D14
D13
D12
10M50SAE144C8G
U8H
BANK8
IO/DEV_CLRn
IO/DEV_OE
IO/VREFB8N0
IO/CONFIG_SEL
IO/nCONFIG
IO/CRC_ERROR
IO/nSTATUS
IO/CONF_DONE
D11
floating
floating
H_C_8
will be used as IO
H_C_9
L8
R29
10k
VCC_3.3F
L7
L6
L5
L4
floating
L3
R31
10k
VCC_3.3F
R32
10k
VCC_3.3F
L2
L1
10M50SAE144C8G
U8I
BANK9
GND
REFGND
VCCIO1A
VCCIO1B
VCCIO2
VCCIO3
VCCIO4
VCCIO5
VCCIO6
VCCIO7
VCCIO8
VCCA1
VCCA2
VCCA3
VCCA4
VCCA5
VCCA6
VCC_ONE
ADC_VREF
ANAIN1
L7 bead
AGND
filter required
VCC_3.3F
C54 0.1uF
VCCA3.3
C57 0.1uF
C58 0.1uF
C55 0.1uF
C56 0.1uF
L8 bead
V133
ADC_F0
10M50SAE144C8G
U8C
BANK6
IO/CLK2p
IO/CLK2n
IO/CLK3p
IO/CLK3n
IO/DPCLK3
IO/VREFB6N0
IO/DPCLK2
CLK50
H_B_17
H_2B_17
H_B_18
H_2B_18
BT1
BT2
BT3
H_C_1
H_C_2
H_C_3
H_C_4
H_C_5
H_C_6
H_C_7
10M50SAE144C8G
Title
FPGA
Size
A4
Number
Revision
Date:
1/11/2018
Sheet of
File:
G:\My Drive\...\FPGA.SchDoc
Drawn By:


VCC_3.3F
GND
C24
0.1uF
floating
U2
AREF
UVcc
V233
VCC
VCC
AVCC
AVCC
L1
L2
1pf
C22
0.1uF
C23
0.1uF
D-
D+
R7 22Ω
R8 22Ω
C21
1uF
VCC_5V
UCap
VBus
RST
RESET
XTAL2
XTAL1
UGnd
GND
GND
GND
GND
Y1
XTAL
C20
20pF
C31
20pF
ATmega16U4-AU
PF0 (ADC0)
PF1 (ADC1)
PF4 (ADC4/TCK)
PF5 (ADC5/TMS))
PF6 (ADC6/TDO
PF7 (ADC7/TDI)
PE6 (INT.6/AIN0)
PE2 (HWB)
PC7 (ICP3/CLK0/OC4A)
PC6 (OC3A/OC4A)
PD7 (T0/OC4D/ADC10)
PD6 (T1/OC4D/ADC9)
PD5 (XCK1/CTS)
PD4 (ICP1/ADC8)
PD3 (TXD1/INT3)
PD2 (RXD1/INT2)
PD1 (SDA/INT1)
PD0 (OC0B/SCL/INT0)
PB7 (PCINT7/OC0A/OC1C/RTS)
PB6 (PCINT6/OC1B/OC4B/ADC13)
PB5 (PCINT5/OC1A/OC4B/ADC12)
PB4 (PCINT4/ADC11)
PB3 (PDO/PCINT3/MISO)
PB2 (PDI/PCINT2/MOSI)
PB1 (PCINT1/SCLK)
PB0 (SS/PCINT0)
WP
L_AD
TCK
TMS
PF6
TD
R3 22Ω TDO
HOLD
PE2
CLKOUT
PC6
CM7
CM6
CM5
CM4
CM3 TXD
CM2 RXD
CM1
CM0
R13 330Ω
INT1
PB5
ACC_CS
MISO
R21 1k MISO
MOSI
SCLK
SS
D15
GND
VCC_3.3F
R15 10k
D11 1N4148
RST
S12
switch
C53 0.1uF
GND
VCC_3.3F
D16 Photodiode
L_AD
R19 330k
GND
Title
Microcontroller
Size A4
Number
Revision
Date: 1/11/2018
Sheet of
File: C:\Users\...\Microcontroller.SchDoc
Drawn By:


P10
CLKOUT
CM7
CM6
CM4
CM5
CM3_TXD
CM2_RXD
CM1
CM0
GND
Header 10
P1
VCC_5V
VCC_3.3A
VCC_3.3F
GND
Header 10
L3
L4
bead
F1
500mA
VUSB
J1
VBUS
D-
D+
GND
440119-1
C4
0.1uF
button_lock
PWR
R1
120Ω
D9
U3 LF33ABDT-TR
IN
OUT
C3
0.1uF
C2
2.2uF
J3
J7
J4
J5
J6
RG2
C15
1uF
C17
1uF
C18
100pF
R16 21k
ADP3334
R18
10k, 1%
PM2
RH
UC
STR
RL
VCC
DC
RW
100k
SW6
SW5
SW_tact
D10
FSV8100V
C19
10nF
C_L
C_H
C2_L
C2_H
V133
V233
J3-
J3+
J8-
J8+
C16
100nF
C34
10nF
C33
0.1uF
C14
0.01uF
CS1
current_sense
VCC_3.3AF
VCC_3.3AF2
C10
100nF
C35
10nF
C36
0.1uF
C11
0.01uF
R44
205Ω
R45
820Ω
U10
THS4302
THS4302RGTR
L12
L5
C37
47pF
C38
0.1uF
C39
22uF
R42
30.1Ω
P14
J9
R43 Header
100Ω
Title
Power
Size
A4
Number
Revision
Date:
1/11/2018
Sheet of
File:
C:\Users\Shuo\Desktop\Power.SchDoc
Drawn By:


R20
DS1
VCC_3.3F
D11
D12
D13
D14
D15
D16
D17
D18
56Ω
DP
Green
Red
L1
L2
L3
L4
L5
L6
L7
L8
R33
R34
R35
R36
R37
R38
R39
R40
47Ω
56Ω
D1
D2
D3
D4
D5
D6
D7
D8
GND
S1
S2
S3
S4
S5
S6
S7
S8
S9
S0
switch
P3
P8
P4
Header
ADC_F0
ADC_F1
ADC_F2
ADC_F3
ADC_F4
ADC_F5
ADC_F6
ADC_F7
ADC_F8
H_A_1
H_A_2
H_A_3
H_A_4
H_A_5
H_A_6
H_A_7
H_A_8
H_A_9
H_B_1
H_B_2
H_B_3
H_B_4
H_B_5
H_B_6
H_B_7
H_B_8
H_B_9
H_B_10
H_B_11
H_2B_12
H_2B_13
H_2B_14
H_2B_15
H_2B_16
H_2B_17
H_2B_18
H_C_1
H_C_2
H_C_3
H_C_4
H_C_5
H_C_6
H_C_7
H_C_8
H_C_9
Y4
ASFL1
opsn
Tri_State
GND
Vdd
Output
CLK50
C29
0.01uF
Title
Peripheral circuits
Size
A4
Number
Revision
Date:
1/11/2018
Sheet of
File:
C:\Users\...Peripheral circuits.SchDoc
Drawn By:


J18
GND
D+
D-
VBUS
D2+
D2-
VUSB
Y2
C1
0.1uF
U4
VDD
OSC1
OSC2
RST
GP7/TxLED
GP6/RxLED
GP5
GP4
GP3
TX
VSS
D+
D-
Vusb
GP0/SSPND
GP1/USBCFG
GP2
CTS
RX
RTS
MCP2200
C32
0.1uF
VCC_3.3F
R22
10k
R25
470Ω
RST4
R26
1k
C7
0.1uF
S10
switch
P11
Header 4X2
U7
RN4870
D14
Yellow
R28
330Ω
SW4
SW_tact
R27
4.7k
P12
Header 8
S11
open_pull_high
P20
R41
4.7k
Title
Bluetooth
Size
A4
Number
Revision
Date: 1/11/2018
Sheet of
File: C:\Users\Shuo\Desktop\Bluetooth.SchDoc
Drawn By:


U8A
BANK1
IO/ADC1IN1 6 ADC_F1
IO/ADC1IN2 7 ADC_F2
IO/ADC1IN3 8 ADC_F3
IO/ADC1IN4 10 ADC_F4
IO/ADC1IN5 11 ADC_F5
IO/ADC1IN6 12 ADC_F6
IO/ADC1IN7 13 ADC_F7
IO/ADC1IN8 14 ADC_F8
do not parallel with IO
IO/JTAGEN 15 JTAGEN_R30 10k VCC_3.3F
IO/TMS 16 TMS
IO/TCK 17 TCK
IO/TDI 18 TDI
IO/TDO 19 TD
IO 20 CM7
IO 21 CM6
IO 22 CM4
IO 23 CM5
max10_144socket
U8B
BANK2
IO/CLK0n 25 CM3_TXD
IO/CLK0p 26 CM3_RXD
IO/CLK1n 27 CM1
IO/CLK1p 28 CLKOUT
IO/VREFB2N0 29 CM0
IO 30 S9
IO/PLL_L_CLKOUTn 32 S8
IO/PLL_L_CLKOUTp 33 S7
max10_144socket
U8C
BANK3
IO 38 S6
IO 39 S5
IO 40 S4
IO 41 S3
IO 42 S2
IO 43 S1
IO 46 H_B_1
IO/VREFB3N0 44 S0
IO 49 H_A_1
IO 50 H_B_2
IO 51 H_A_2
IO/CLK6n 52 H_B_3
IO/CLK6p 53 H_A_3
IO/CLK7n 55 H_B_4
IO/CLK7p
max10_144socket
U8D
BANK4
IO/VREFB4N0 57 H_A_4
IO 58 H_B_5
IO 60 H_A_5
IO 61 H_B_6
IO 62 H_A_6
IO 63 H_B_7
IO 64 H_A_7
IO 65 H_B_8
IO 66 H_A_8
IO 69 H_B_9
IO 70 H_A_9
max10_144socket
U8E
BANK5
IO/RUP 76 H_B_12
IO 74 H_B_10
IO/RDN 77 H_2B_12
IO 75 H_B_11
IO 82 H_B_14
IO 80 H_B_13
IO 83 H_2B_14
IO/VREFB5N0 81 H_2B_13
IO 85 H_2B_15
IO 84 H_B_15
IO 87 H_2B_16
IO 86 H_B_16
max10_144socket
U8F
BANK6
IO/CLK2p 88 CLK50
IO/CLK2n 89 H_B_17
IO/CLK3p 90 H_2B_17
IO/CLK3n 91 H_B_18
IO 92 H_2B_18
IO 93 BT1
IO/DPCLK3 96 BT2
IO/VREFB6N0 97 BT3
IO/DPCLK2 98 H_C_1
IO 99 H_C_2
IO 100 H_C_3
IO 101 H_C_4
IO 102 H_C_5
IO 105 H_C_6
IO 106 H_C_7
max10_144socket
U8G
BANK7
IO 110 D18
IO 111 D17
IO/VREFB7N0 116 D16
IO 117 D15
IO 118 D14
IO 119 D13
IO 120 D12
max10_144socket
U8H
BANK8
IO 123 D11
IO/DEV_CLRn 124 floating
IO/DEV_OE 125 floating
IO/VREFB8N0 126 H_C_8
IO/CONFIG_SEL 128 will be used as IO H_C_9
IO 127 L8
IO/nCONFIG 130 R29 10k VCC_3.3F
IO 129 L7
IO 131 L6
IO 132 L5
IO 133 L4
IO/CRC_ERROR 134 floating
IO 135 L3
IO/nSTATUS 136 R31 10k VCC_3.3F
IO/CONF_DONE 138 R32 10k VCC_3.3F
IO 140 L2
IO 141 L1
max10_144socket
U8I
BANK9
GND 79 GND
GND 95
GND 68
GND 59
GND 56
GND 48
GND 142
GND 137
GND 121
GND 115
GND 112
GND 104
L7 bead
REFGND 4 AGND
VCCIO1A 9 filter required VCC_3.3F
VCCIO1B 24 VCC_3.3F
VCCIO2 31
VCCIO3 54
VCCIO3 45
VCCIO3 67
VCCIO4 78
VCCIO5 94
VCCIO6 103
VCCIO6 113
VCCIO7 139
VCCIO8 122
VCCIO8 35
C54 0.1uF
GND
VCC_3.3F
L8 bead
VCCA1 34
VCCA2 107
VCCA3 143
VCCA4 71
VCCA5 2 VCCA3.3
VCCA6
C57 0.1uF
C58 0.1uF
C55 0.1uF
C56 0.1uF
GND
VCC_ONE 73 V133
VCC_ONE 72
VCC_ONE 47
VCC_ONE 37
VCC_ONE 36
VCC_ONE 144
VCC_ONE 114
VCC_ONE 109
VCC_ONE 108
VCC_ONE 1
ADC_VREF 5 VCC_3.3F
ANAIN1 3 ADC_F0
GND 145
GND
max10_144socket
Title FPGA socket
Size A4
Number
Revision
Date: 2/2/2018
Sheet of
File: Sheet1.SchDoc
Drawn By:

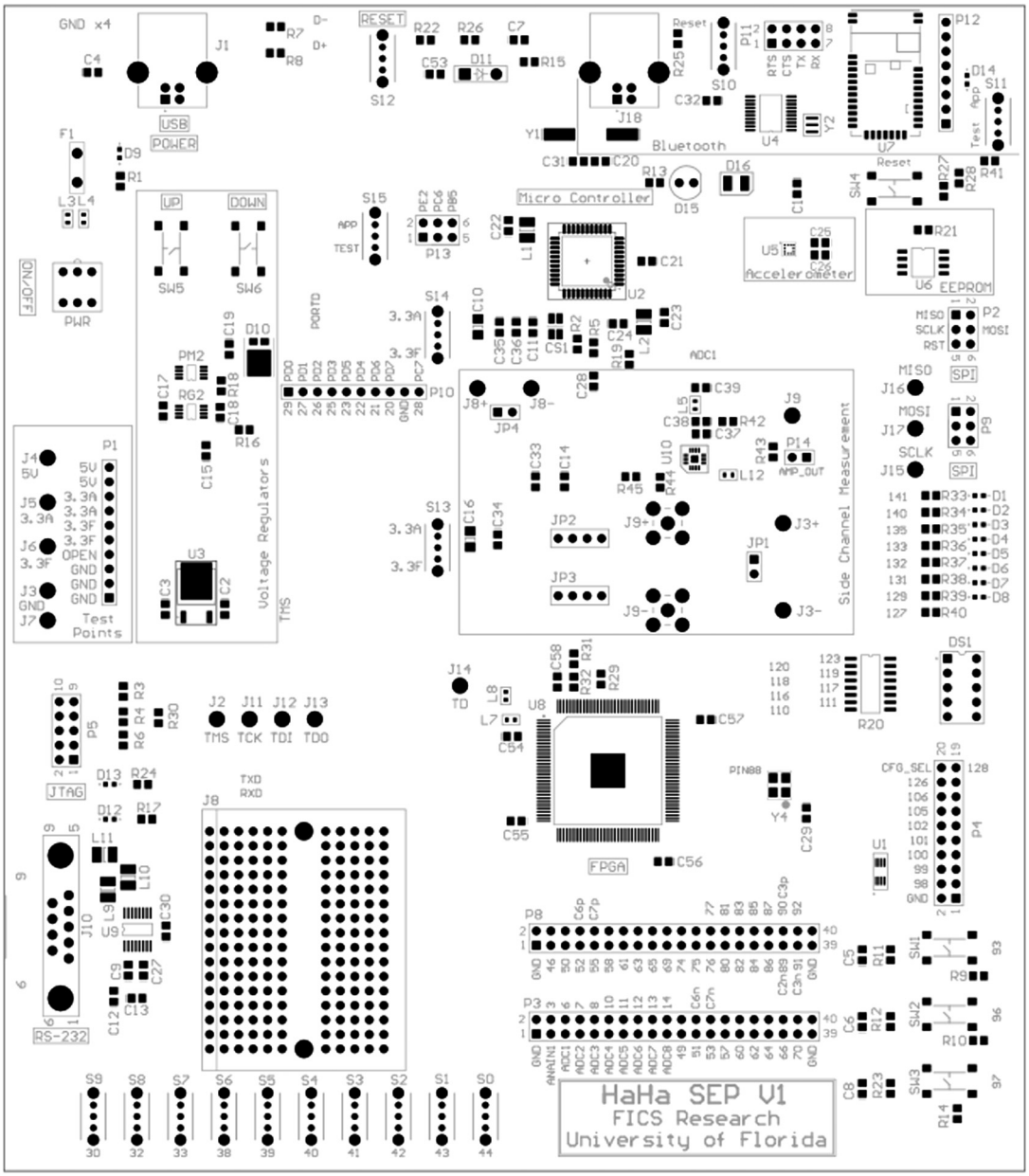


FIGURE A.20

Assembly diagrams of the HaHa board.

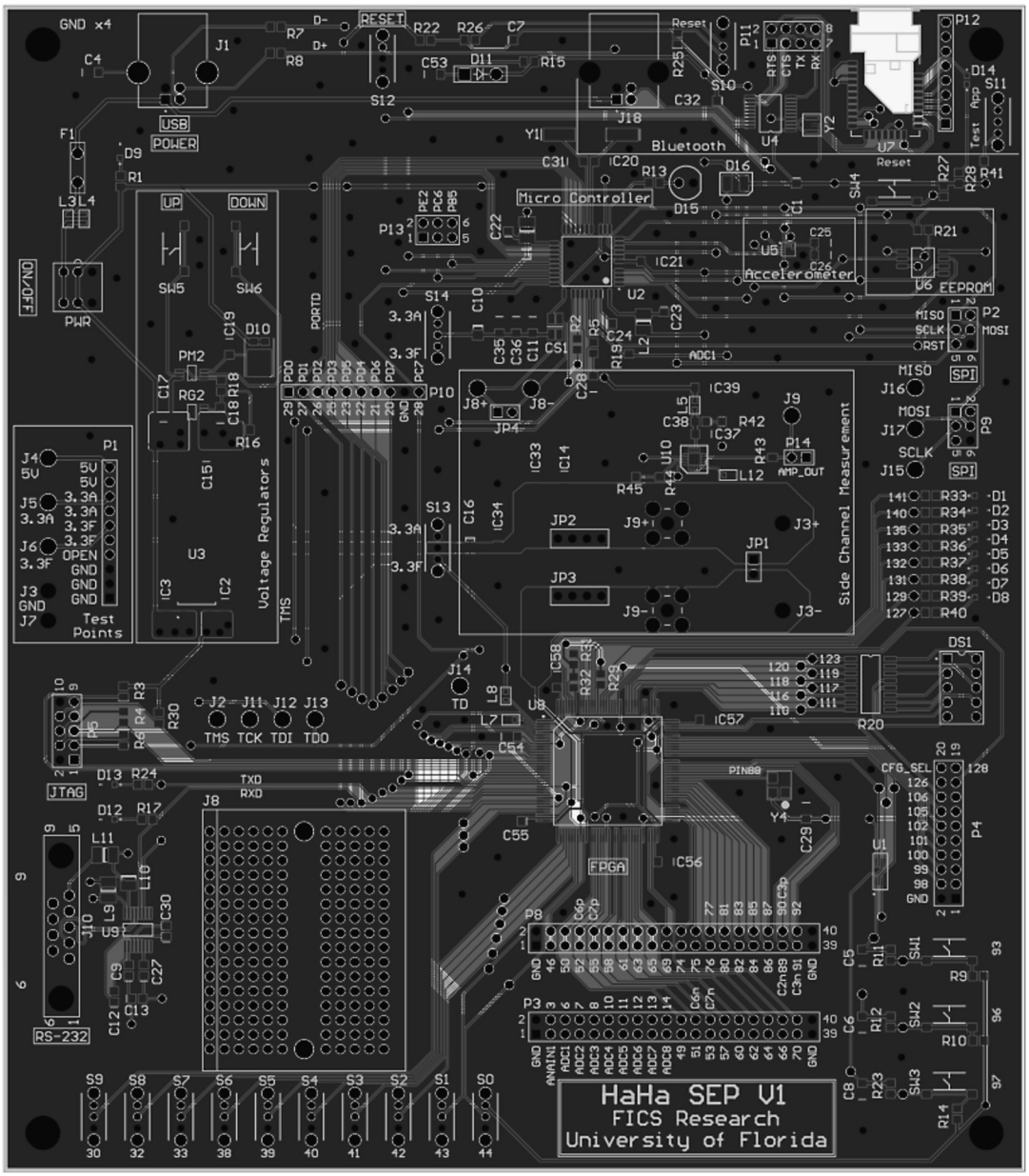


FIGURE A.21

Layout of the HaHa board.