Final Project Report
CPE 224: Digital System Design Laboratory

Cuckoo

A simple FPGA + IoT digital alarm clock

CODUSTRY

Team members

Kompipat Kanpisan #57070503403
Nutchanon Ninyawee #57070503410
Thunpisit Amnuaikiatloet #57070503418

Class Instructor

Mr.Sanan Srakaew

Department of Computer Engineering
King Mongkut's University of Technology Thonburi

126 Pracha-U-Thit Rd, Tungkru, Bandmod., Bangkok 10140

Find out more about the project on

codustry.com/cuckoo & github.com/codustry/cuckoo

Requirements

Our requirement is to create a project using 400,000 gates FPGA Xilinx's based logic device. Project knowledge have to related to CPE224: Digital System Design class.

Backgrounds

We want to continue our latest digital project which is digital alarm clock but we want to push it further.

We decide to made a research on human behavior and what comes out is creating a simple alarm clock that mix with well-known traditional clock "Cuckoo". But hanging alarm clock that just beeping and set by mechanic or button is too simple for us.

We take it further by decide that these day it is IoT (internet of things) era. We should make this clock to be set-able and control-able from the internet. So user can set time and dismiss alarm from anywhere in the world. Also, time is automatically calibrated when it's connected to the internet.

Concept

From our requirements that we have to use FPGA to create gates and show time in digital we decide to add internet to the clock by using IoT (internet of things) micro controller board called "Particle Photon".

Particle Photon connect FPGA to the internet. Particle itself can be programmed over the internet so basically we can program FPGA over the internet.

Equipments usage

- APEX : FPGA D3 XC3S200
Programmable 400,000 gates Xilinx's based logic device [SN: 28]

- Particle : Photon
An Arduino's based internet of things micro controller. [SN: PH-150808-R7JX-0]

- UNI-T : UT33 series
Palm Size Digital Multimeters [SN: 1110565695]

- Hakko : FX-888D
Professional soldering station set [SN: 05888312053134]

- **Codustry's X Station**
Intel Core i7, 16 GB DDR4, 500 GB SSD with GTX970 Graphics card.

- Glue gun, Hot glue, Arcylics sheet, Wooden sheet, LED (red, green), Jumping wires, Soldering lead, Screw & bolts, Codustry's duck

Softwares usage

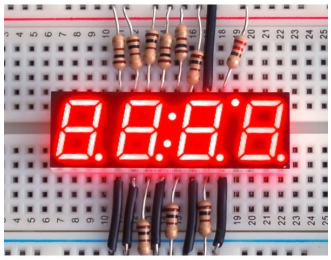
- ISE Design Suite 14.7
A software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.

- Particle build & dashboard
Particle Build IDE is an always-online, browser-based portal where your Photon code can be edited, saved, and shared. This cloud-based IDE handles code compilation and flashes the built binaries to your Photon over-the-air. You don't even need your Photon next to you to update its program.

- Rhinoceros 3D
Rhino can create, edit, analyze, document, render, animate, and translate NURBS* curves, surfaces, and solids, point clouds, and polygon meshes. There are no limits on complexity, degree, or size beyond those of your hardware.

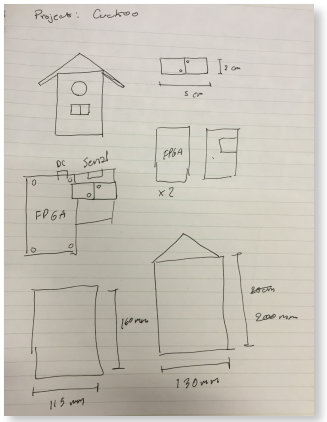
Designs & Hardwares

+



=

Figure 1.1 Figure 1.2 Figure 1.3



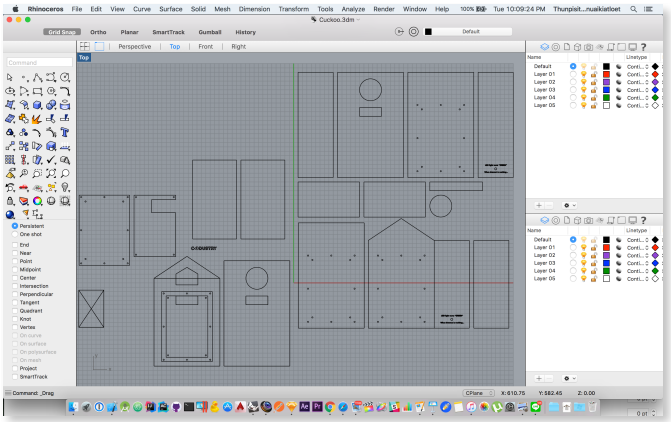


Figure 1.4 Figure 1.5

Ideas & concept

We have an idea of combining traditional Cuckoo wall-hang clock (Figure 1.1) with modern (Figure 1.2) and electronics looks. The concept is to make it look sleek and simple as possible with no human interaction to the clock directly because everything must be controlled over internet. We started by making a research on type of Cuckoo clock and how does it look in modern looks. The right one is the final look of the Cuckoo clock (Figure 1.3)

Sketch to production

Then I sketch up the design of the clock and measurement of the designed Cuckoo clock (Figure 1.4). Then I draw a cad design in Rhinoceros. There were 2 designs one is basic rectangle clock and another which is production one is Cuckoo style clock. I then later use laser cut machine to cut the wood out and stick them together.

Electronics & Softwares

We've uploaded our code both for FPGA in Verilog HDL language & Particle Phonon in Arduino language on to Github. To view full code please visits: https://github.com/codustry/cuckoo

Experiences with FPGA

Spartan 3 is not that pleasant due to many issues that caused by Xilinx and the board itself. Installing Xilinx on Windows 10 is never a good idea. There are many bugs and errors that should be fixed. There is another funny thing about the board we got. The board is physically version 200, but when it's plugged into computer, a firmware in the board version 400. Because of this, all ports shown in both data sheets are wrong and I had to test all the ports to see which one is which. But finally I'm get used to this and every get fixed, it is going to be fun programming it.

Jobs of FPGA

FPGA board is consisted of arrays of transistors that are not configured (unwired). Xilinx let programmers to wire the resistors by just programming their behavior or even to assign logic gates manually. This enables us to create a prototype of chip and rewiring it anyway we want. For example in our project, we do not even know how to wire transistors together to create a counter, but we just write Verilog code in behavioral style and let Xilinx translate it and wire for us.

Experiences with Particle Photon

It is intuitive and handy to use photon. Writing, Compiling, flashing to the target device are never easier than this. I did it online with the web app. The platform also provided a "dashboard". It contains all information of devices we owned at glance. We could sending debug messages to the dashboard. In the old traditional way, we need to send those data back to serial port.

Jobs of Particle Photon

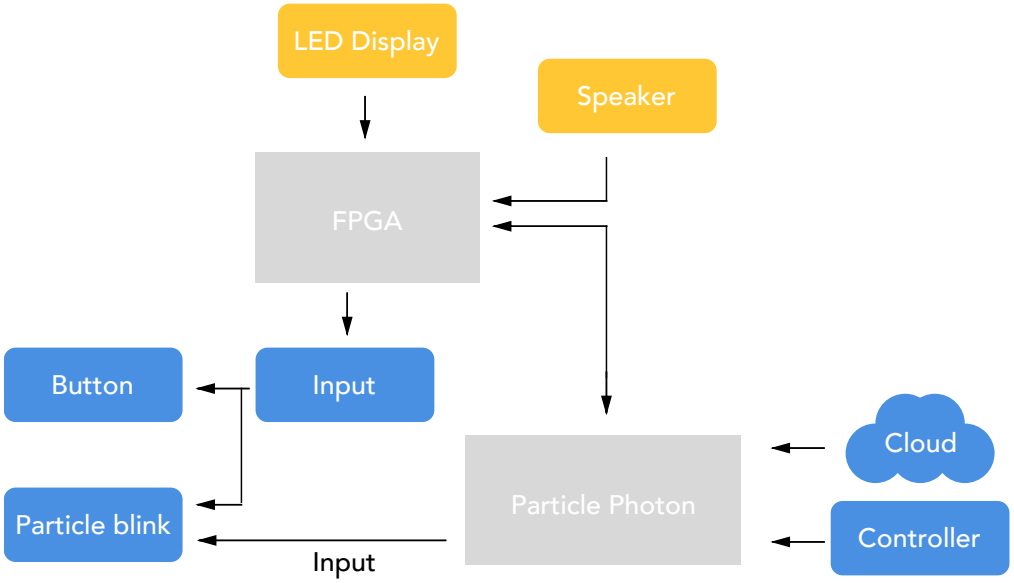
The photon device acts as an automatic time updater using NTP server ,user interface and a bit like controller. We don't want it to do FPGA jobs even through it could be done. User can interact with the clock using only api of particle.io , IFTT or command line. Here is the list of interactions.

1. force ringing
2. set timezone
2. change ringtone
3. force update time
4. set alarm
5. turn on-off alarm
6. knowing the state of FPGA such as alarm ringing or alarm never set.

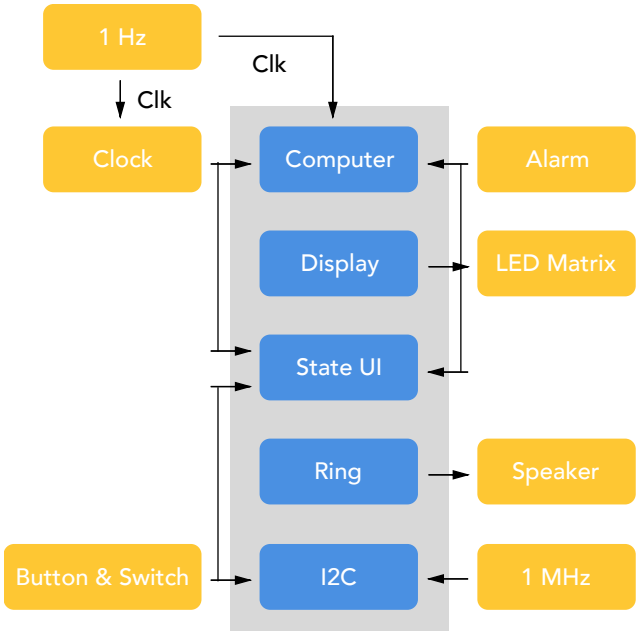
All of those are not available without internet. But user still can set the time using the buttons on the board. Updating time is done using a NTP server. The default timezone is GMT+7. And it is keeping update time every hour.

Diagrams

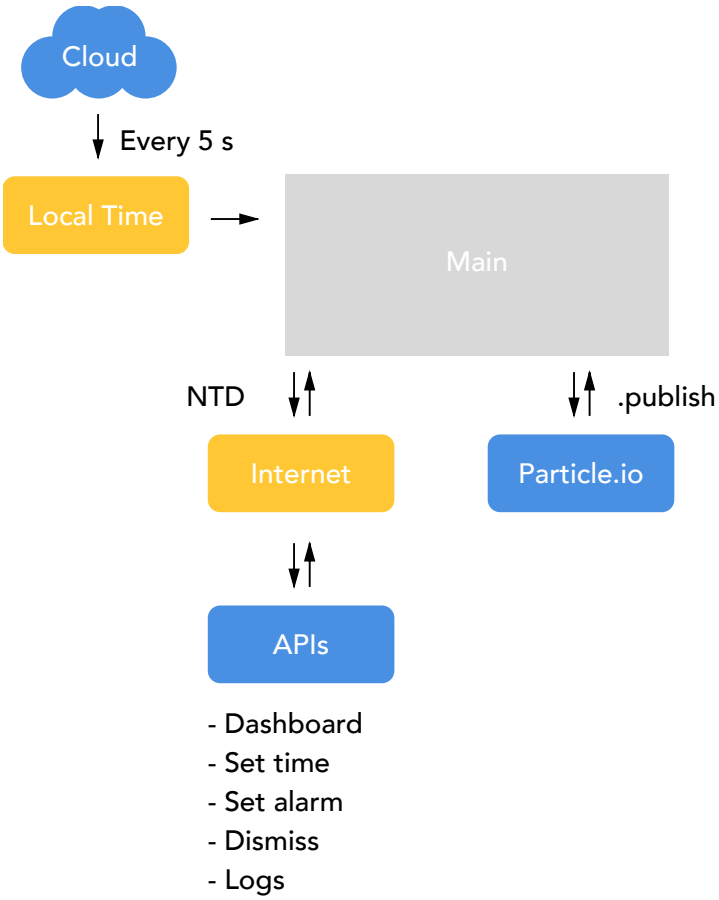
1.Whole system



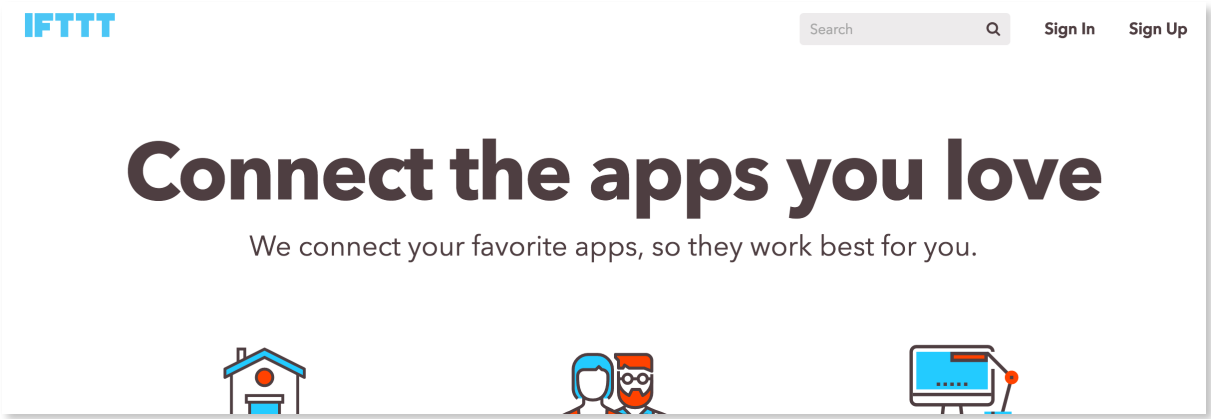
2.FPGA Module



3.Particle Photon Module



User Interface



For user interface part since we are using internet of things method we would like users to interact with the Cuckoo directly from the internet. We use the best alternative that we can find which is using **IFTTT**, is a free web-based service that allows users to create chains of simple conditional statements, called "recipes", which are triggered based on changes to other web services such as Gmail, Facebook, Instagram, and Pinterest. **IFTTT** is an abbreviation of "If This Then That".

We connect to the Cuckoo and use **IFTTT** developed mobile app called "**DO Button**" which creates button that can be push to activate command. These screenshots below represent a demo command of the Cuckoo from DO Button mobile application to Cuckoo.

Examples

I'm in Bangkok - DO Button send command to Particle Photon to find "bangkok" in the function and see that the timezone is +7 so it set the time for Bangkok then it send out command to FPGA in binary to tells FPGA that change the time from whatever it is to Bangkok. (Figure 3.1)

Set alarm at 9:00 - DO Button send command to Particle Photo that it wants to set alarm to "9:00" and then it find function that set command. Then it send out the commands to FPGA in binary to set the alarm from whatever it is to 9:00 (Figure 3.2)

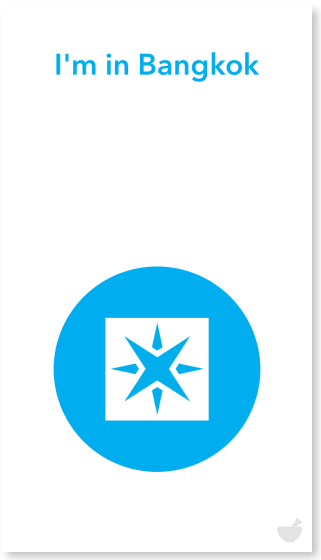


Figure 3.1

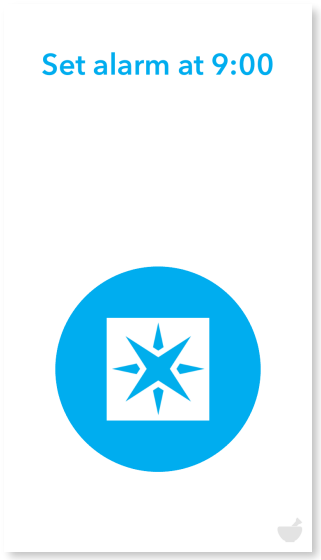


Figure 3.2

File - Cuckoo.ino

A set of code use to compile and upload to Particle Photon device. Language is based on Arduino.
To view full and up to date code please visits: https://github.com/codustry/cuckoo

```
// This #include statement was automatically added by the Particle IDE.
#include "SparkTime/SparkTime.h"

#include <climits>
#define bitRead(value, bit) (((value) >> (bit)) & 0x01)
#define bitSet(value, bit) ((value) |= (1UL << (bit)))
#define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
#define bitWrite(value, bit, bitvalue) (bitvalue ? bitSet(value, bit) : bitClear(value, bit))
#define bit(b) (1UL << (b))

#define PI 3.14159265358979323846264338327950
#define HALF_PI 1.57079632679489661923132169163975
#define TWO_PI 6.28318530717958647692528676655901
#define DEG_TO_RAD 0.017453292519943295769236907684886
#define RAD_TO_DEG 57.295779513082320876798154814105
typedef uint16_t word;
#define abs(x) ((x)>0?(x):-(x))
#define radians(deg) ((deg)*DEG_TO_RAD)
#define degrees(rad) ((rad)*RAD_TO_DEG)
#define sq(x) ((x)*(x))

#define lowByte(w) ((uint8_t) ((w) & 0xff))
#define highByte(w) ((uint8_t) ((w) >> 8))

UDP UDPClient;
SparkTime rtc;
unsigned long currentTime;
unsigned long lastTime = 0UL;
String timeStr;
int timezone = 0;

const byte numPins = 8;
byte pins[] = {0,1,2,3,4,5,6,7};
bool sw[] = {false, true, false, false};

bool alarmOnceSet = false;
bool timeOnceSet = false;
int alarmHour = 0;
int alarmMinute = 0;
int isRinging = 0xf;

void setup() {
    Particle.publish("devStatus", "Device ON - cuckoo activate!");

    Particle.function("setAlarm", setAlarmOnFPGA);
    Particle.function("updateTime", setTimeOnFPGA);
    Particle.function("ringASecond", ringAFewSecond);
    Particle.function("triggerRing", triggerRing);
    Particle.function("setTimezone", setTimezone);
    Particle.function("setCTimezone", setTestTimeZone);
    Particle.variable("alarmHour", alarmHour);
    Particle.variable("alarmMinute", alarmMinute);
```

```
    Particle.variable("isRinging", isRinging);
    Particle.variable("timezone", timezone);

    //internet time sync
    rtc.begin(&UDPClient, "north-america.pool.ntp.org");
    rtc.setUseEuroDSTRule(true);
    rtc.setUseDST(false);
    rtc.setTimeZone(7); // gmt offset


    //set pin D0-D7 to be output
    for(int i=0;i<numPins;i++){
        pinMode(i, OUTPUT);
    }
    //setOpPins(0x8);
    //setDataPins(0xf);

    //set pin A0-A3 to be input
    for(int i=10;i<14;i++){
        pinMode(i, INPUT);
    }
    setIdleStateOnFPGA();

    //set debug LED
    pinMode(A4, OUTPUT); //FGPA Programmed?
    pinMode(A5, OUTPUT); //internet?

    delay(1000);
    reportState();
}

void loop() {
   /* if(isRinging != 0xf){
        if(timeOnceSet == true){
            //check if an hour already
        }else{
            setTimeOnFPGA();
        }
    }
    delay(1000*5);
    reportState();*/
}

// for GMT -7 use -7 but for GMT +7 use 6
//eg (GMT+,DST) -8
int setTimezone(String timezoneArg){
    int commaIndex = timezoneArg.indexOf(',');

    int gmt = timezoneArg.substring(0,commaIndex).toInt();
    String dst = timezoneArg.substring(commaIndex);
    if(dst == "true"){
        rtc.setUseDST(true);
    }else{
        rtc.setUseDST(false);
    }
    rtc.setTimeZone(gmt);
    setTimeOnFPGA("");
    Particle.publish("debug", "run setTimezone, tz="+String(gmt)+" dst="+dst);
```

```
    return 1;
}

int setTestTimeZone(String place){
    //Asia
    if(place == "bangkok"){
        setTimezone("+7,false,false");
        Particle.publish("Set Timezone","I'm in Bangkok");
    }
    else if(place == "shanghai"){
        setTimezone("+8,false,false");
        Particle.publish("Set Timezone","I'm in Shanghai");
    }
     else if(place == "beijing"){
        setTimezone("+8,false,false");
        Particle.publish("Set Timezone","I'm in Beijing");
    }
     else if(place == "singapore"){
        setTimezone("+8,false,false");
        Particle.publish("Set Timezone","I'm in Singapore");
    }
     else if(place == "taipei"){
        setTimezone("+8,false,false");
        Particle.publish("Set Timezone","I'm in Taipei");
    }
     else if(place == "tokyo"){
        setTimezone("+9,false,false");
        Particle.publish("Set Timezone","I'm in Tokyo");
    }
    else if(place == "seoul"){
        setTimezone("+9,false,false");
        Particle.publish("Set Timezone","I'm in Seoul");
    }
    //Europe
    else if(place == "london"){
        setTimezone("0,false,true");
        Particle.publish("Set Timezone","I'm in London");
    }
    else if(place == "paris"){
        setTimezone("+2,false,true");
        Particle.publish("Set Timezone","I'm in Paris");
    }
    else if(place == "zurich"){
        setTimezone("+2,false,true");
        Particle.publish("Set Timezone","I'm in Zurich");
    }
    else if(place == "berlin"){
        setTimezone("+2,false,true");
        Particle.publish("Set Timezone","I'm in Berlin");
    }
    else if(place == "rome"){
        setTimezone("+2,false,true");
        Particle.publish("Set Timezone","I'm in Rome");
    }
    else if(place == "moscow"){
        setTimezone("+3,false,false");
        Particle.publish("Set Timezone","I'm in Moscow");
    }
```

```
    //America
    else if(place == "newyork"){
        setTimezone("-4,true,false");
        Particle.publish("Set Timezone","I'm in NewYork");
    }
    else if(place == "sanfrancisco"){
        setTimezone("-8,true,false");
        Particle.publish("Set Timezone","I'm in San Francisco");
    }
    else if(place == "honolulu"){
        setTimezone("-10,true,false");
        Particle.publish("Set Timezone","I'm in Honolulu");
    }
    else if(place == "denver"){
        setTimezone("-6,true,false");
        Particle.publish("Set Timezone","I'm in Denver");
    }
    else if(place == "miami"){
        setTimezone("-4,true,false");
        Particle.publish("Set Timezone","I'm in Miami");
    }
    //Africa
    else if(place == "capetown"){
        setTimezone("+2,false,false");
        Particle.publish("Set Timezone","I'm in Cape Town");
    }
    else if(place == "dubai"){
        setTimezone("+4,false,false");
        Particle.publish("Set Timezone","I'm in Dubai");
    }
    //Else
    else{
        setTimezone("0,false,false");
        Particle.publish("Set Timezone","Default Timezone to 0 GMT");
    }
}


void checkInternet(){
    if(WiFi.ready())digitalWrite(A5, HIGH);
    else digitalWrite(A5, LOW);
}

void reportState(){
    checkInternet();
    short state = 0;
    for(int i=0;i<4;i++){
        bitWrite(state, i, digitalRead(10+i));
    }
    digitalWrite(A4,LOW);
    switch (state) {
        case 0: Particle.publish("fpgaStatus", "normal"); break;
        case 1: Particle.publish("fpgaStatus", "never Set Clock"); break;
        case 2: Particle.publish("fpgaStatus", "never Set Alarm"); break;
        case 3: Particle.publish("fpgaStatus", "ringing"); break;
        case 0xf: Particle.publish("fpgaStatus", "fpga is on but unprogramed"); digitalWrite(A4,HIGH);
break;
        default: break;
```

```
    }
    isRinging = state ;
}

void setIdleStateOnFPGA(){
    setOpPins(0x8);
    setDataPins(0);
    delay(100);
}

int setTimeOnFPGA(String dummy){
    Particle.publish("debug", "run setTimeOnFPGA");
    currentTime = rtc.now();

    short hour =  rtc.hour(currentTime);
    short hour_ = hour/10;
    short hour__ = hour-hour_*10;
    setOpPins(0xf);//1111 tenth
    delay(50);
    setDataPins(hour_);
    delay(100);

    setOpPins(0xe);//1110 unit
    delay(50);
    setDataPins(hour__);
    delay(100);

    Particle.publish("debug", "h="+String(hour_)+String(hour__));
    setIdleStateOnFPGA();

    short min =  rtc.minute(currentTime);
    short min_ = min/10;
    short min__ = min-min_*10;
    setOpPins(0xd);
    setDataPins(min_);
    delay(10);

    setOpPins(0xc); //1100
    setDataPins(min__);
    delay(10);

    setIdleStateOnFPGA();

    short second =  rtc.second(currentTime);
    short second_ = second/10;
    short second__ = second-second_*10;

    setOpPins(0xb);
    setDataPins(second_);
    delay(10);

    setOpPins(0xa); //1010
    setDataPins(second__);
    delay(10);
    setIdleStateOnFPGA();

    setIdleStateOnFPGA();
    Particle.publish("Debug", "Clock is seted at "+rtc.ISODateString(currentTime));
```

```
    return 1;
}

int setAlarmOnFPGA(String alarmTime){
    Particle.publish("debug", "run setAlarmOnFPGA with arg="+alarmTime);

    alarmHour = alarmTime.substring(0,2).toInt();
    short hour_ = alarmHour/10;
    short hour__ = alarmHour-hour_*10;
    setOpPins(0x6);
    setDataPins(hour__);
    delay(10);

    setOpPins(0x7);
    setDataPins(hour_);
    delay(100);
    setIdleStateOnFPGA();

    alarmMinute = alarmTime.substring(3).toInt();
    short min_ = alarmMinute/10;
    short min__ = alarmMinute-min_*10;
    setOpPins(0x4);
    setDataPins(min__);
    delay(10);
    setOpPins(0x5);
    setDataPins(min_);
    delay(10);

    Particle.publish("debug", "it took h="+alarmTime.substring(0,2)+"m="+alarmTime.substring(3));
    setIdleStateOnFPGA();
    Particle.publish("Debug", "Alarm is seted at "+String(alarmHour)+":"+String(alarmMinute));
    return 1;
}

int setAlarmNActive(String alarmTime){
    Particle.publish("debug", "run setAlarmOnFPGA with arg="+alarmTime);

    alarmHour = alarmTime.substring(0,2).toInt();
    short hour_ = alarmHour/10;
    short hour__ = alarmHour-hour_*10;
    setOpPins(0x6);
    setDataPins(hour__);
    setOpPins(0x7);
    setDataPins(hour_);
    setIdleStateOnFPGA();

    alarmMinute = alarmTime.substring(3).toInt();
    short min_ = alarmMinute/10;
    short min__ = alarmMinute-min_*10;
    setOpPins(0x4);
    setDataPins(min__);
    setOpPins(0x5);
    setDataPins(min_);
    setIdleStateOnFPGA();

    turnAlarm("on");

    Particle.publish("debug", "it took h="+alarmTime.substring(0,2)+"m="+alarmTime.substring(3));
```

```
    Particle.publish("Debug", "Alarm is seted at "+String(alarmHour)+":"+String(alarmMinute));
    return 1;
}



int turnAlarm(String onOrOff){
    if(onOrOff=="on"){
        setOpPins(0x8);
        sw[1]=true;
        updateSW();
    }else{
        setOpPins(0x8);
        sw[1]=false;
        updateSW();
    }

    setIdleStateOnFPGA();
    return 1;
}

int triggerRing(String dummy){
    if(sw[2]){
        setOpPins(0x8);
        sw[2]=false;
        updateSW();
    }else{
        setOpPins(0x8);
        sw[2]=true;
        updateSW();
    }

    setIdleStateOnFPGA();
    return 1;
}

int ringAFewSecond(String dummy){
    setOpPins(0x0);
    sw[3]=true;
    sw[1]=true;
    updateSW();
    delay(3000);
    sw[1]=false;
    sw[3]=false;
    updateSW();
    delay(100);
    sw[3]=true;
    updateSW();
    setIdleStateOnFPGA();

    return 1;
}

/*function to set pin D0-D3 to desire byte*/
void setDataPins(byte data){
    for(int i=0;i<4;i++){
        byte state = bitRead(data, i);
```

```
        digitalWrite(pins[i], state);
        delay(100);
    }
}

/*function to set pin D0-D3 to desire byte*/
void updateSW(){
    for(int i=0;i<4;i++){
        byte state = bitRead(sw[i], 0);

        digitalWrite(pins[i], state);
        delay(100);
    }
}

/*function to set pin D4-D7 to desire number*/
void setOpPins(byte data){
    for(int i=7;i>3;i--){
        byte state = bitRead(data, i-4);
        digitalWrite(pins[i], state);
    }
}
```

File - Display.v

A main code which call sub-modules and programmed FPGA device. To view full and up to date code please visits: https://github.com/codustry/cuckoo

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    18:36:19 04/19/2016
// Design Name:
// Module Name:    display
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module display(
    output [6:0] seg,
            output [3:0] segEn, status,
            output [7:0] led,
            output dp,buzz, isRinging,
            //input [7:0] ci,//charlesInterface
    input sysclk, setH, setM, displayMode, onOffAlarm, yudRongSW,
            input [1:0] k,
            input [7:0] photon

    );
            parameter h = 1'b1;
            wire sigDot1s,sig4ms,sig1hz, displayModeW, setHW, setMW, forceRingW, dismissW,
onOffAlarmW, isSetting;


            wire [1:0] sigWhichDigit;
            wire [3:0] valDigit,out;

            wire setSH, setSM, setSaH, setSaM;
            wire [3:0] m_,m__,h_,h__,
                                                aM_, aM__, aH_, aH__,
                                                cH_,cH__,cM_,cM__;

            wire[5:0] isSyncClock;
            wire[3:0] isSyncAlarm;
            wire[3:0] photonData;



            assign dp = (sigWhichDigit[1]^sigWhichDigit[0])&sig1hz&(displayMode);
```

```
	assign {h_, h__, m_, m__} = (displayMode)? {cH_,cH__,cM_,cM__}:{aH_, aH__, aM_, aM__};

	assign {cSetH,cSetM} = (displayMode)? {(setH),(setM)}: {h,h};
	assign {aSetH,aSetM} = (displayMode)?  {h,h}: {(setH),(setM)};

	dataSel1_4 valSwap(valDigit, sigWhichDigit, h_, h__, m_, m__);
	counter_2bit swapDigit(sigWhichDigit, sig4ms);
	clk_1s tick(sigDot1s,sysclk);
	clk_4ms tick2(sig4ms,sysclk);
	BCDto7seg valDisplaySig(seg, valDigit);
	decoder2_4 SsegSwap(segEn, sigWhichDigit);

	cl charlesInterface(isSyncClock, isSyncAlarm, photonData, displayModeW, onOffAlarmW,
forceRingW, setHW, setMW, dissmissW, photon);

	clock
molClk(cH_,cH__,cM_,cM__,sig1hz,sigDot1s,cSetH,cSetM,isSyncClock,photonData,photonData,photonDa
ta,photonData,photonData,photonData);
	alarm molAlarm(aH_, aH__, aM_, aM__, aSetM, aSetH,
sigDot1s,isSyncAlarm,photonData,photonData,photonData,photonData);

	/* reg isSetting;
	reg [1:0] timePosition,tmp;*/
	assign isSameTime = {cH_,cH__,cM_,cM__} == {aH_, aH__, aM_, aM__};
//	always @(negedge forceRingW) begin
//		yudRongIoT <= 0;
//	end
	cuckooRinging ring(buzz, isRinging, (isSameTime&(!isSetting))|forceRingW, sigDot1s,
yudRongSW &dissmissW , onOffAlarm & (!forceRingW) & onOffAlarmW);

	//check whether user is interact hhmm setting
	isSettingFn checkIsSetting(isSetting, cM__[0], setM, setH);
	/*initial tmp =3;
	always @( setM ,setH,timePosition) begin
		if (!(setM&setH)) begin
			tmp <= timePosition;
			isSetting <=1;
		end else begin
			if(tmp == timePosition) isSetting <=1;
			else begin isSetting <=0;tmp <= 3;			end
		end
	end
	always @(posedge cM__[0]) begin
			if(timePosition==2) timePosition<=0;
			else timePosition<=timePosition+1;
	end*/

	//i2c slave
	wire [7:0] linkData;
	I2CslaveWith8bitsIO link(SDA,SCL,linkData);

	//debug zone
	assign led = {photon};
	assign status = 4'b1100;
endmodule
```