Project Design Architecture

Joeseph Sande

CST-451 Capstone Project Final Architecture & Design

Grand Canyon University

Instructor: Professor Mark Reha

Revision: Initial Draft

Date: 11-20-22

ABSTRACT

The Wakup9000 will be a digital clock that will keep track of appointments for the user, light up when it is time for set meetings, display the time on the digital clock, and display a message relating to what status the clock is in at that time. This device will be perfect for users who desire to keep a physical clock in their home offices to remind them when their appointments are. When the clock sets off an alarm, it will light up the LED lights on the board and display a message on the LCD Display screen for the user while flashing the time on the board while it still continues to move the clock's time forward.

With FPGA technology at the team's disposal, this is a perfect task for them to tackle. The hardware contains all of the features that this alarm clock requires and it is configurable in the field if the client were to want additional features in the future or for current features to be manipulated. This leads to a potential stream of revenue for the company in the future as we continue to make updates to the product and send our employees out for field operations to update purchased products for the clients. An example of a future addition to this product would be to create an outside application that can connect to the alarm clock to automatically update the product and sync up with the user's calendar. For the Wakup9000's current proposed features, the clock will conduct all of the required features of displaying a message to the user, displaying the time, allowing the user to utilize the various inputs to configure the clock settings and turn off the alarm, and flash LED lights when the alarm goes off.

History and Signoff Sheet

Change Record

Date	Author	Revision Notes
11-18-22	Joeseph	Initial draft for review/discussion

Overall Instructor Feedback/Comments

Overall Instructor Feedback/Comments

Integrated Instructor Feedback into Project Documentation

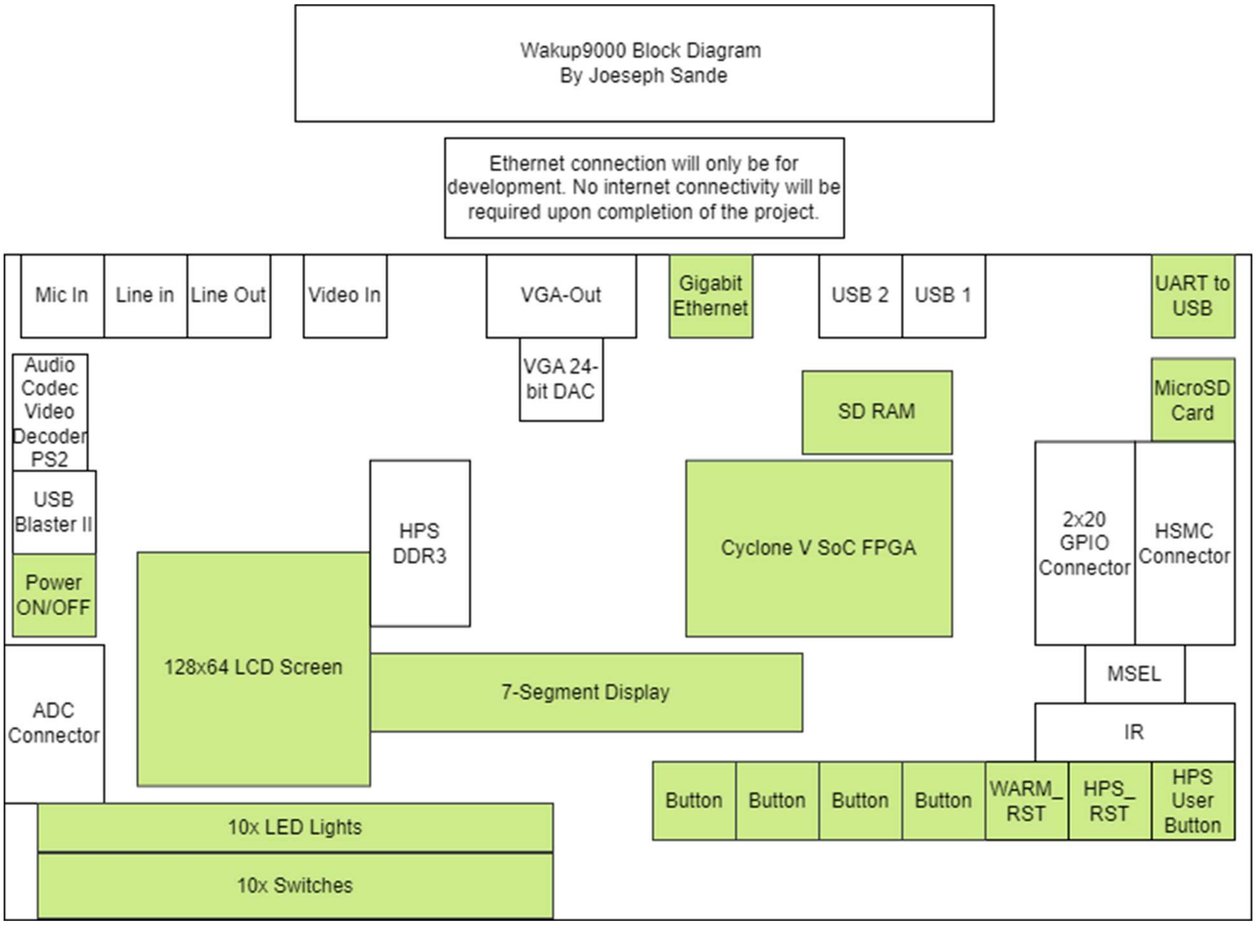
☐ Yes ☐ No

TABLE OF CONTENTS

Design Introduction

The Wakeup9000 requires the ability to wake the heaviest sleeper from their naps and will provide the ability for the team to gain familiarity with FPGA products to implement more complex solutions to future problems. This project will be the team's first interaction with imbedded systems, so this is a perfect challenge for them. The DE-10 Standard FPGA Board has every component required to create a functional clock that will cover all of the required specifications of waking up a heavy sleeping individual. The necessary components that will be utilized to achieve this goal are the buttons, switches, lights, 7-segment display for the time, and the LCD screen. Below is a high-level diagram of the components being utilized for this project.

Block Diagram of High-Level Solution



Components Utilized:

1. LCD Screen – will be utilized to display clock status messages to the user.
2. 5x switches – will be utilized to configure settings for the clock.
3. 7x buttons – will be utilized to configure settings for the clock.
4. 10x LED lights – will light up when the alarm goes off.
5. 7-Segment Display – will be used to display the time of the clock using a BCD decoder.
6. MicroSD Card Slot – will be used to hold the program files for the clock.
7. Gigabit Ethernet port – will be used to connect the clock to a computer for development using FileZilla data transfers.
8. UART to USB port – will be used to connect the board to a development environment.
9. USB Port – is optional for a WIFI adapter, but will not be used in this project.
10. Cyclone V SoC FPGA Processor- will be used to process the application.
11. SD RAM- will be used to process the application.

High-Level Solution Description

The solution to creating the Wakup9000 will incorporate the components within the board on the above diagram. This board runs on an ARM processor that will be programmed to incorporate the components listed above. This board will utilize its buttons and switches for providing the user with the ability to configure the board's settings, to turn off, and to reset the alarm. The Cyclone V SoC FPGA Processor will process general requests being sent to the board. The 128x64 LCD Screen will display the messages for what the alarm is relating to for the user. The MicroSD Card slot will hold the MicroSD card that contains the Linux and program files for the board to operate. The 7-segment display will be where the time is shown on the board. The use of the Gigabit Ethernet or a WIFI adapter to a USB slot on the board is optional to provide internet capabilities for the board and will only be required for development purposes. Once the connection is established, the program can be run on the board to initialize the clock from the preset time of midnight. The clock will then become configurable by the user when they toggle the configuration switch for either the base clock configuration mode or the alarm configuration mode. The buttons designated for seconds, minutes, and hours will accept input to increase each respective time component by one from each input by the user. This input will have to be saved utilizing the save button or cancelled by utilizing the exit button before the configuration mode switch is toggled off. If the switch is toggled off, the settings will be lost and it will continue incrementing the time as if the user never entered into the configuration mode.

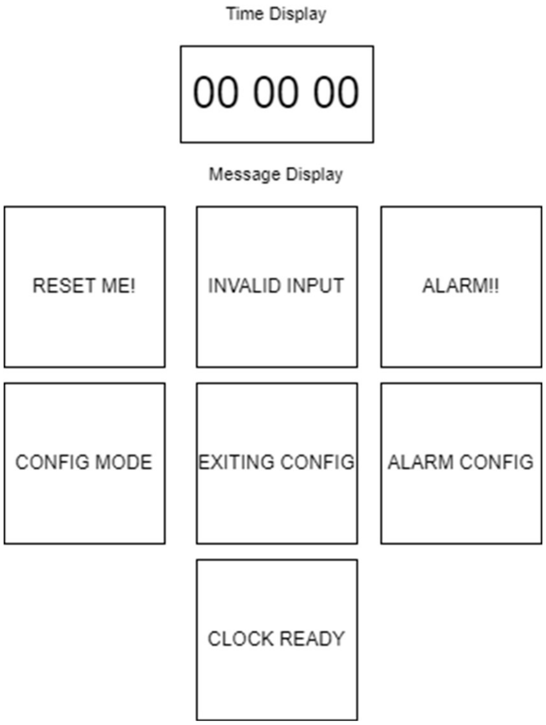
External Reference Chart

ID	Deliverable Description	Comments	Evaluator (internal or external as applicable)	Status	Date of Decision
1	Terasic, (2018), "DE10-Standard User manual," pages 1-134	Utilized for general familiarization and architecture of the board.	External	Approved	10/28/2022
2	MainClockFlowchart.png	Main clock flow of operations and processes.	Internal	Approved	10/28/2022
3	AlarmConfigFlowchart.png	Alarm clock configuration process.	Internal	Approved	10/28/2022
4	AlarmStatusFlowchart.png	Alarm status process.	Internal	Approved	10/28/2022
5	ClockConfigFlowchart.png	Clock configuration process.	Internal	Approved	10/28/2022
6	Terasic, (2017), "DE10-Standard_Computer_ARM"	Utilized for finding memory allocation for components on the board.	External	Approved	10/28/2022

Detailed High-Level Solution Design

Functional Design

Wireframe Diagram:



Proof of Concepts		
Description	**Rationale**	**Results**
1. Program LED lights on FPGA Board.	Utilization of LED lights for the clock.	Understanding on how to program LED lights has been achieved.
2 – Program LCD on FPGA Board.	Utilization of LCD screen for the clock.	Understanding on how to program the LCD screen has been achieved.
3 – Program buttons on FPGA Board.	Utilization of buttons for the clock.	Understanding on how to program buttons has been achieved.
4 – Program switches on FPGA Board.	Utilization of switches for the clock.	Understanding on how to program switches has been achieved.
5 – Program 7-segment display on FPGA Board.	Utilization of a time for the clock.	Understanding on how to program a time through a 7-segment display has been achieved.
7 – Program FPGA in Go Lang (**Out-of-Scope**)	Integration of new language in place of C in ARM processing.	Research indicates this is possible, but it is currently out of scope.

Hardware and Software Technologies or Tools	Justification
ARM Eclipse IDE v.22.3	**The development environment for creating the FPGA programs.**
SD Memory Card Formatter v.5.0.2	**To format an SD card for the clock.**
Win32 Disk Imager v.1.0.0	**To format the SD card to run Linux.**
DE10-Standard Linux SD Card Image v.1.3.0	**To run Linux on the clock.**
DE10-Standard-UP-Linux v.1.3.0	**To run Linux on the clock.**
DE10-Standard FPGA Board	**The main hardware being manipulated into a clock.**
FileZilla v.3.60.2	**Used in development to transfer files to the board.**
PuTTY v.77	**Used in development to test and run files on the board.**
Cygwin v.3.3.6	**Used in development to run Linux commands on Windows.**
GDB v.7.4	**Used for debugging in the development environment.**
MinGW v.11.2.0	**Used for the development environment for board communication through Windows OS.**
Linaro toolchain v.4.8	**Linux kernel, GNU Compiler Collection (GCC), QEMU, power management, graphics, and multimedia interface for ARM instructions.**

Physical and Logical Solution Design:

Design Description:

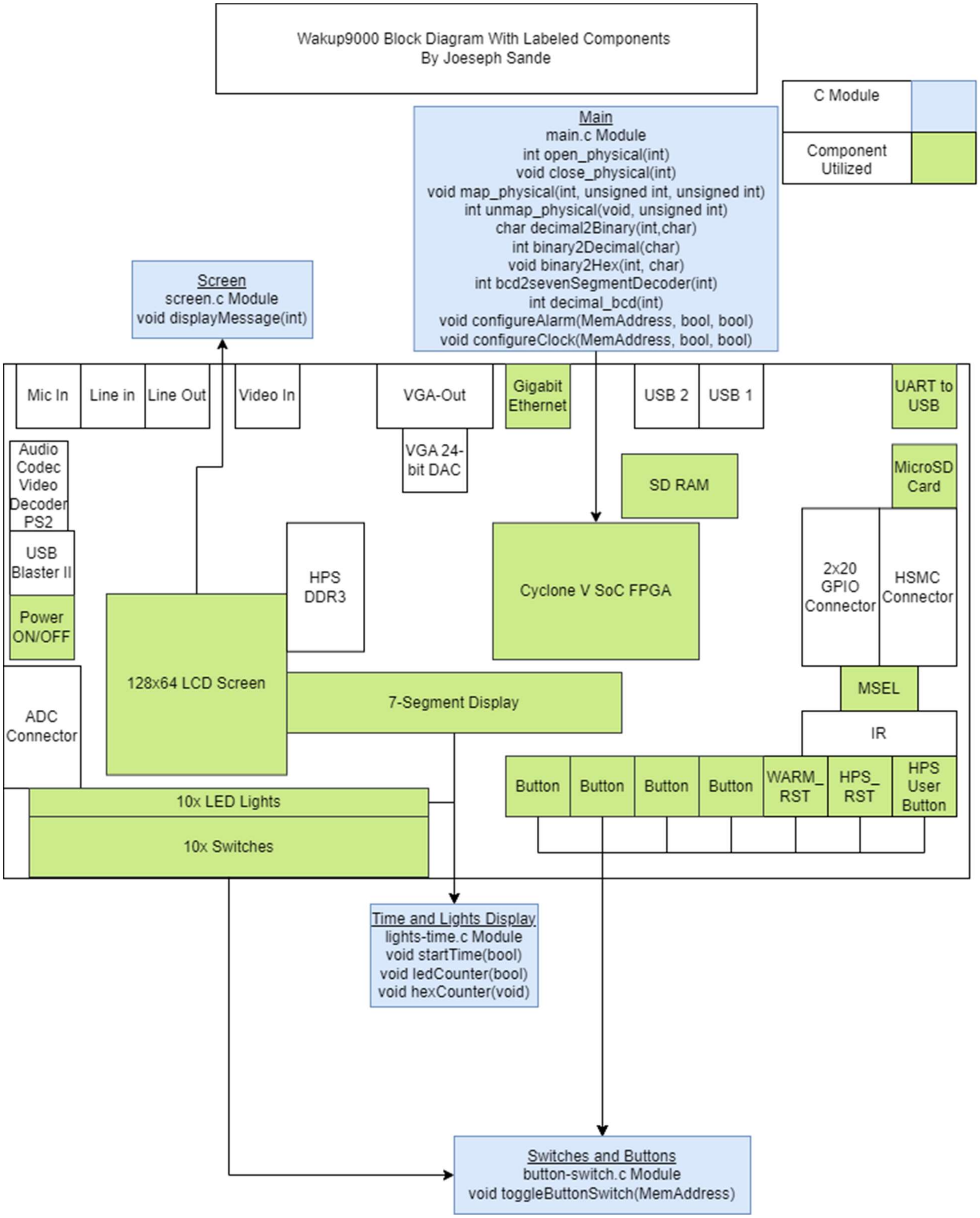
The Wakeup9000 will be created utilizing four C modules consisting of the button-switch, screen, time-lights, and main modules. The button-switch module will control the memory addresses and operations associated with the buttons and the switches on the clock. The lights-time module will control the memory addresses and operations associated with the time display and the lights on the clock. The main module will control the general logic flow of the application for the clock and will utilize all of the other modules for the clock to operate the specified components. This module will contain the loops that run the processes imported from the respective modules for the clock to display the time, display messages, process input from buttons, process input from switches, continuously check for the clock's state to set the alarm off when the alarm time is reached, and allow configurations to be saved from the user's input.

Libraries Included in Project:

1. Stdio.h
2. Unistd.h
3. Fcntl.h
4. Font.h
5. Sys/mman.h
6. Pthread.h
7. Address_map_arm.h
8. Stdint.h
9. Stdlib.h
10. Terasic_os_includes.h
11. Terasic_lib.h
12. LCD_Lib.h
13. LCD_Hw.h
14. LCD_Driver.h
15. LCD_Lib.h
16. Lcd_graphic.h
17. Font.h
18. Linux/kernel.h
19. Linux/module.h
20. Linux/init.h
21. Linux/interrupt.h
22. Asm/io.h
23. Address_map_arm.h
24. Socal/socal.h
25. Socal/hps.h
26. Socal/alt_gpio.h

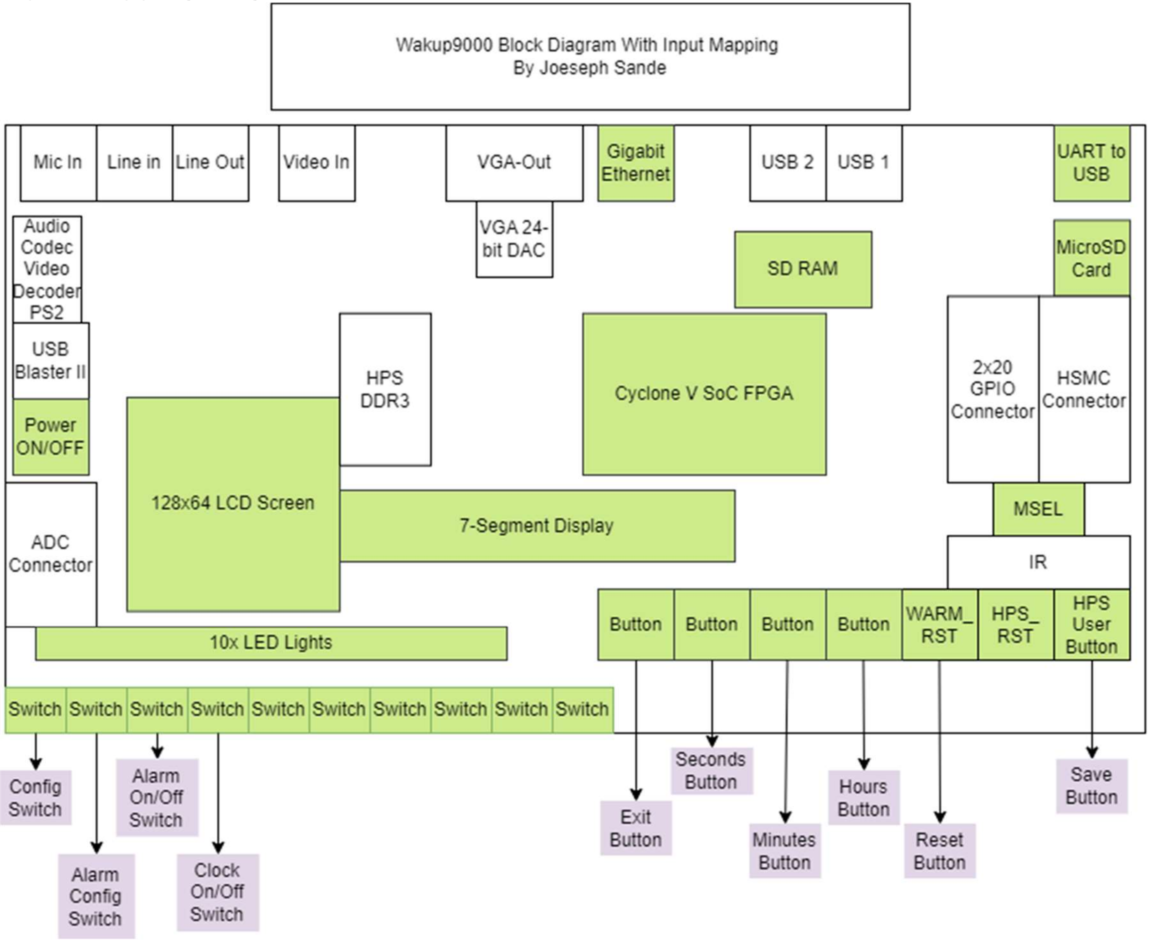
Key Technical Design Decisions:

Design Block Diagram:



The key technical design decisions taken to accomplish the task of designing the Wakeup9000 utilizes a modular-based project structure using the C programming language. This project is broken up into four modules to support the major general features between those modules. The modules will be button-switch.c, lights-time.c, main.c, and screen.c. Each of these modules are named after the type of components that they will be controlling on the clock and will each have a massive impact on the project without making the main.c file too large to maintain its code in the future. A proof of concept has been completed for each of the clock's components for functionality utilizing the tools outlined in this project, so these operations have been completely confirmed to be operational without posing harsh risks for the project. The button-switch.c module will contain the method necessary for manipulation of each button or switch memory address. The lights-time.c module will contain the methods necessary for starting the clock's time, counting up the time, and incrementing through the LED lights when the alarm goes off. The screen.c module will contain the method necessary for manipulating the screen's display for each device status change. The main.c module will hold the heart of the application with the conversion methods, memory mapping, clock configuration, alarm configuration, and initiating the main variables for the clock to operate.

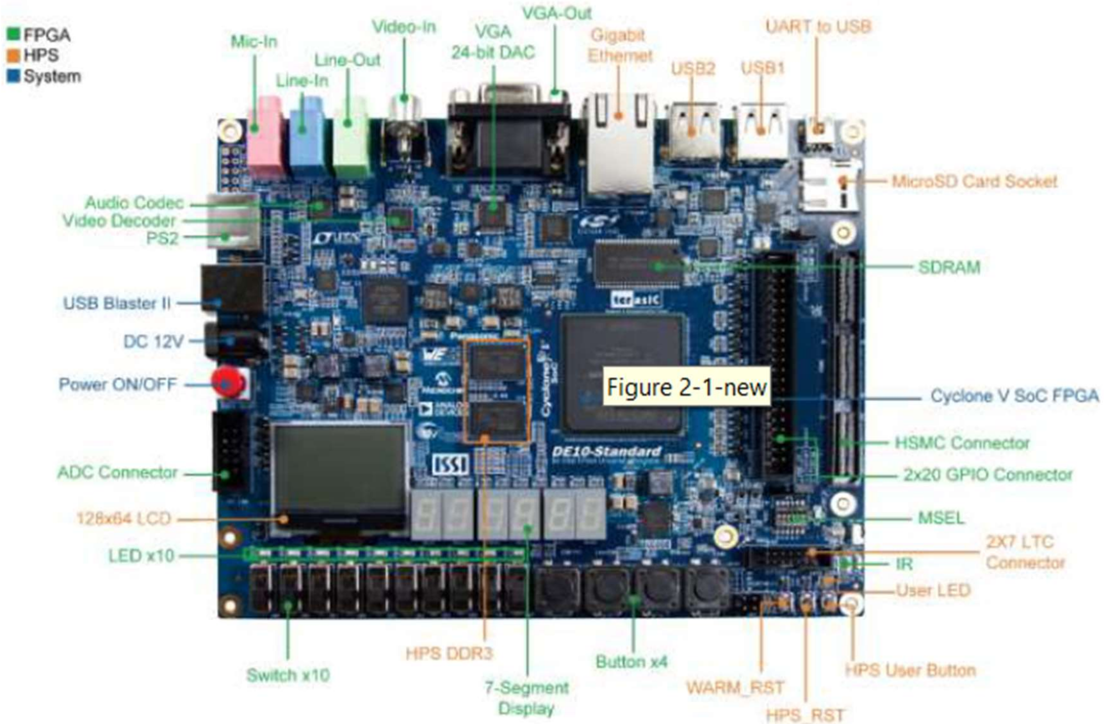
Input Mapping Diagram:



The buttons and switches being utilized have been labeled in the diagram above with four switches being utilized for the clock's configuration regarding the time, alarm configuration, turning the alarm on and off, and turning the clock on and off. The buttons being utilized have been labeled as the exit button, seconds button, minutes button, hours button, reset button, and save button for the clock. These buttons are associated with their named functions of exiting the operation without saving for the exit button, adjusting the seconds on the time display, adjusting the minutes on the time display, adjusting the hours on the time display, saving configurations, and resetting the clock.

Detailed Component Diagram:

This diagram was utilized from Terasic's, "DE10-Standard User manual," page 6.



The diagram above was pulled from the DE-10 Standard FGPA Board manual from Terasic and shows every component of the board being utilized in this project in a detailed perspective. There are many other components that are not being utilized in this diagram, but the main components that are being utilized were directly addressed in the previous diagram. This board has great potential for many device implementations.

Detailed Technical Design

General Technical Approach:

The general approach taken to accomplish the task of designing the Wakeup9000 will be implemented by avoiding the use of interrupts for ensuring that the clock's operations function continuously until the user toggles the configuration or alarm configuration switches. When inputs are toggled during the clock's operations, they will send their process to their own designated module to determine what operation the input will trigger. The implementation of these processes will utilize loops in order to sustain numerous operations.

Flow Charts/Process Flows:

Please see attached MainClockFlowchart.pdf that shows the surface level logic flow for the clock and its general processes and for if the user interacts with buttons where their input is not being recorded in the current state of the clock, the inputs will be processed as null to avoid conflicts with the clock's operations. This flow begins at the clock being turned on by the user and proceeds through the user's interactions with the components on the clock. Please see ClockConfigFlowchart.pdf for the case of if the user toggles the configuration or alarm configuration switches, the clock will enter into or leave out of configuration mode. This diagram shows the process flow for the clock when a user is configuring the clock's time. Please see the AlarmStatusFlowchart.pdf for the flow of operations for the clock with a set alarm by the user. This diagram shows the process flow for the clock as it continuously checks for an active alarm's time so that it can activate the designated components when the alarm time is reached. Please see AlarmConfigFlowchart.pdf for if the user interacts with the alarm config switch to enter into or leave alarm configuration mode. This diagram shows the process flow for when a user wants to set or adjust the alarm time for the clock's alarm. Please see the ClockStatusFlowchart.pdf for the continuous check the clock will conduct on itself to detect which state it is in for which message it should display on the LCD screen. This process will change the clock's message displayed for the user to know what the current status of the clock is for both error detection and notifications for whether their intended operations are entering into or exiting out of the right modes correctly.

Functions Pseudo Code by Module

Button-Switch Module:

```
/*
* toggleButtonSwitch takes a memory address of either a button or a switch to manipulate the data
* to if it is toggled or not.
* Return type = void
```

```
*/
void toggleButtonSwitch(MemoryAddress activated button or switch memory address){
        //switch statement to activate or deactivate the button or switch from 0 to 9
        switch (activated button or switch memory address) {
            //cases for each switch/button number 0 to # of buttons and switches
            case memory address: {
                switch or button logic to manipulate associated memory addresses
                break;
            default {
                //This occurs when the memory address is not found
                LCD screen displays “INVALID INPUT”
                break;
            }
        }
    }
}
```

Lights-Time Module:

```
/*
*Method to start ticking the clock’s time based upon a Boolean variable being true
* Return type = void
*/
Void startTime(bool for if clock is on) {
    //start clock incrementation while clock is on
    While(bool for if clock is on) {
        //initialize variables
```

```
HEX pointer1 = 0;

HEX pointer2 = 0;

Int incrementer = 0;

Int minutes = 0;

Int hours = 0;

//for loop to increment the time display values

for (int x) {

    //convert x value into bcd value

    Int value = (decimal_bcd((conversion logic for x variable));

    *HEX pointer1 = value;

    //Seconds and minutes up to 59 so that it does not go above 60

    If (HEX pointer1 > 59){

        minutes = minutes + 100;

    }

    //same for minutes

    If (minutes == 5900){

        Hours = hours + 10000;

    }

    //hours not to exceed 23

    If (hours = 240000){

        Hours = 0;

        Minutes = 0;

    }

    HEX pointer1 = 0 + minutes + hours;

    //increment by one for the incrementor variable

    ++Incrementor;
```

```
            //convert the other HEX pointer to its binary coded decimal value
            HEX pointer2 = decimal_bcd(decimal_bcd((conversion logic for
            Incrementor));
        }
    }
}
/*
*Method to increment the LED lights
* Return type = void
*/
void ledCounter(bool alarmGoingOff) {
    //when the alarm is going off, start the while loop to increment the LEDs
    While(alarmGoingOff) {
        *LED pointer = 0;
        //Increment through each LED to light them up and then sleep for a second
        for (int i){
            //for loop will assign the LED pointer variable to the i incrementor for the for
            //loop
            LED pointer = i;
            Sleep for 1 second;
        }
    }
}
/*
*Method to increment the time on 7-segmentDisplay
* Return type = void
```

```
*/
void hexCounter(void) {
        /*begin while loop to count the numbers up starting from 0 to 60. When one hits 60, the screen
        *higher will increment by one until it hits 60, and then the third display on the left will increment
        *to 23 before turning back to 0 to show hours.
        */
        While (1)
        HEX_ptr1 = 0;
        HEX_ptr2 = 0;
        //begin for loop to convert values for duration of while loop
        for (int x…){
                //convert values for display through the decimal_bcd method
                int value = (decimal_bcd((x/1000)%10)<<24) | (decimal_bcd((x/1000)%10)<<16) |
                (decimal_bcd((x/1000)%10)<<8) | decimal_bcd(x/%000);
                HEX_ptr1 = value;
                //Sleep for one second
                usleep(1*100000);
        }
}
```

Screen Module:

```
/*
*Method to display messages in accordance with which status the device is in
* Return type = void
*/
```

```
void displayMessage(int deviceStatus){
    /*
    *switch statement for deviceStatus to display messages in accordance with which status the
    *device is in
    */
    switch(deviceStatus) {
        // for configure status
        Case 1:
        Screen display = “CONFIG MODE”;
        break;
        //for alarm configure status
        Case 2:
        Screen display = “ALARM CONFIG”;
        break;
        //for critical error status
        Case 3:
        Screen display = “RESET ME!!”;
        break;
        // for active alarm status
        Case 4:
        Screen display = “ALARM!!”;
        break;
        // for exiting configuration mode status
        Case 5:
        Screen display = “EXITING CONFIG”;
        break;
```

```
                // for invalid input status
                Case 6:
                Screen display = “INVALID INPUT”;
                break;
                // default case
                default:
                Screen display = “CLOCK READY”
                break;
        }
}
/*
*Main module to control all operations of the logic by implementing the other three modules through its
*main process.
* Return type = void
*/
```

Main Module:

```
//Switches memory address variables
Switch0MemoryAddress;
Switch1MemoryAddress;
Switch2MemoryAddress;
Switch3MemoryAddress;
Switch4MemoryAddress;
//Buttons memory address variables
Button0MemoryAddress;
Button1MemoryAddress;
Button2MemoryAddress;
```

```
Button3MemoryAddress;

Button4MemoryAddress;

//Time display memory address variables

HEX_ptr1;

HEX_ptr2;

//LED lights memory address variables

LEDR_ptr;

//LCD Screen memory address variables

HW_REGS_BASE;

HW_REGS_SPAN;

HW_REGS_MASK;

USER_IO_DIR;

BIT_LED;

BUTTON_MASK;

//Clock’s check for running and status variables

bool running = true;

int deviceStatus = 0;

/*

*Main to run program through main module

* Return type = void

*/

void main(){

    // Create virtual memory access to the FPGA light-weight bridge

    if ((fd = open_physical (fd)) == -1)

      return (-1);

    if ((LW_virtual = map_physical (fd, LW_BRIDGE_BASE, LW_BRIDGE_SPAN)) == NULL)
```

```
  return (-1);
//while loop for when the clock is functioning properly to perform the different processes within
//the program
While(running) {
    //Establish the processes for the clock
    //Time process
    startTime(void);
    //Alarm configure
    If(clockAlarmConfig) {
        While(clockAlarmConfig) {
            configureAlarm(MemAddress, bool, bool);
        }
    }
    //Clock configure
    If(clockConfig) {
        While(clockConfig) {
            configureClock(MemAddress, bool, bool);
        }
    }
    //Screen process
    displayMessage(status);
    //Buttons or switches toggle
    toggleButtonSwitch();
    //Turn clock off check
    if (clockOffButton > 0) {
        //method to shut off the clock if the off button is pressed by the user
```

```
            shutOffClock();
            running = false;
        }
    }
}
/*
*method to configure the time for the alarm to trigger
* Return type = void
*/
void configureAlarm(buttonMemAddress button, bool clockIsOn, bool inConfigMode){
    //check to see if the clock is running and if the configure alarm button was toggled
    if (button != 0 && clockIsOn){
        //if seconds button pressed increment seconds by one
        if (button1MemAddress != inactivated button 1 memory address) {
            //add 1 to the pointer to increment to the next second
            HEX_ptr1 = HEX_ptr1 + 1;
        }
        //if minutes button pressed increment minutes by one
        If (button2MemAddress != inactivated button 2 memory address) {
            //add 60 to pointer to increment to the next minute
            HEX_ptr1 = HEX_ptr1 + 60;
        }
        //if hours button pressed increment hours by one
        if (button3MemAddress != inactivated button 3 memory address) {
            //add 1 to address pointer to set it to the next hour
            HEX_ptr2 = HEX_ptr2 + 1;
        }
    }
    //else to catch invalid inputs
    else {
        //print a statement telling the user that they need to change what they are doing
        printf(“Doing nothing because you messed up”);
        ScreenMemAddress == display (“INVALID INPUT”);
    }
}
/*
* Method to convert decimal numbers to the binary-coded decimal number
* Return type = Int
*/
Int decimal_bcd(int decimal){
    //switch case to convert decimal value for binary-coded decimal value for device communication
    Switch (decimal) {
        //binary coded decimal conversions for decimal values
        case 0:
            return 0x3f;
        case 1:
            return 0x06;
        case 2:
            return 0x5b;
        case 3:
            return 0x4f;
        case 4:
            return 0x66;
        case 5:
            return 0x6d;
        case 6:
            return 0x7d;
        case 7:
            return 0x07;
        case 8:
            return 0x7f;
        case 9:
            return 0x67;
        default:
            return 0xff;
    }
}
/*
* Method to convert binary numbers to decimal numbers
* Return type = Int
*/
Int binary2Decimal(char input[]){
    //create three integer values
    Int output = 0;
    Int power = 1;
    Int x = strlen(intput) – 1;
    //while loop to go through input char array
    While(x>=0){
        Output = output + ((input[x]-48 * power);

        Power = power *2;

        --x;

    }

    return output;

}


/*

* Method to convert binary numbers to hexadecimal

* Return type = Int

*/

Void binary2Hex(int input, char characters[]){

    char digit[2] = {‘0’,’\0’};

    int nubble = strlen(output);

    //While loop for each nibble in the input integer

    while (nibble > 0) {

        sprint(&digit[0], “%1x”, input & 0x0F);

        output[nibble-1] = digit[0];

        --nibble;

        input = input >> 4;

    }

}

/*

* Method to convert binary-coded decimal number to seven segment decoder values

* Return type = int

*/
```

```
Int bcd2sevenSegmentDecoder(int value){

        int A = output[0] - 48;

        int B = output[1] - 48;

        int C = output[2] - 48;

        int D = output[3] - 48;

        int a = (~B * ~D) + C + (B * D) + A;

        int b = ~B + (~C * ~D) + (C * D);

        int c = ~C + D + B;

        int d = (~B * ~D) + (~B * C) + (B * ~C * D) + (C * ~D) + A;

        int e = (~B * ~D) + (C * ~D);

        int f = (~C * ~D) + (B * ~C) + (B * ~D) + A;

        int g = (~B * C) + (B * ~C) + A + (B * ~D);

        char values[9];

        values[0] = 48;

        values[1] = g + 48;

        values[2] = f + 48;

        values[3] = e + 48;

        values[4] = d + 48;

        values[5] = c + 48;

        values[6] = b + 48;

        values[7] = a + 48;

        values[8] = 0;

        return (int)binary2Decimal(values);

}

/*

* Method to map physical memory
* Return type = void
*/
Void map_physical(int fd, unsigned int base, unsigned int span){
        void *virtual_base;
        // Get a mapping from physical addresses to virtual addresses
         virtual_base = mmap (NULL, span, (PROT_READ | PROT_WRITE), MAP_SHARED, fd,
        base);
        //if there is an error in mapping the memory return null
        if (virtual_base == MAP_FAILED)
        {
            printf ("ERROR: mmap() failed...\n");
            close (fd);
            return (NULL);
        }
        //if no error occurs, return virtual_base value
        return virtual_base;
}
/*
* Method to close physical memory
* Return type = void
*/
void close_physical(int fd){
        close(fd);
}
/*
* Method to close physical memory
* Return type = void
*/
void open_physical(int fd){
        if (fd == -1) {
                if ((fd = open(“/dev/mem”, (O_RDWR | O_SYNC))) == -1) {
                        printf ("ERROR: could not open \"/dev/mem\"...\n");
                        return (-1);
                }
        return fd;
        }
}
/*
* Method to unmap physical memory
* Return type = Int
*/
Int unmap_physical(void, unsigned int x){
        //if there is an error in removing the mapping of the memory return -1
        if (munmap (virtual_base, span) != 0)
        {
                printf ("ERROR: munmap() failed...\n");
                return (-1);
        }
        return 0;
}
/*
* Method to configure the clock’s time
* Return type = void
*/
Void configureClock(MemAddress button, bool clockIsOn, bool clockConfig){
//check to see if the clock is running and if the configure alarm button was toggled
    if (button != 0 && clockIsOn){
        //if seconds button pressed increment seconds by one
        if (button1MemAddress != inactivated button 1 memory address) {
            //add 1 to the pointer to increment to the next second
            HEX_ptr1 = HEX_ptr1 + 1;
        }
        //if minutes button pressed increment minutes by one
        If (button2MemAddress != inactivated button 2 memory address) {
            //add 60 to pointer to increment to the next minute
            HEX_ptr1 = HEX_ptr1 + 60;
        }
        //if hours button pressed increment hours by one
        if (button3MemAddress != inactivated button 3 memory address) {
            //add 1 to address pointer to set it to the next hour
            HEX_ptr2 = HEX_ptr2 + 1;
        }
        if (button5MemAddress != inactivated button 5 memory address){
            //save configurations for the clock
        }
    }
    //else to catch invalid inputs
    else {
        //print a statement telling the user that they need to change what they are doing
        printf("Doing nothing because you messed up");
        ScreenMemAddress == display ("INVALID INPUT");
    }
}
```

NFR Design:

Accuracy:

The accuracy of the clock will be measured by how close the time keeps track of the set time in relation to the ticking of real time. The clock’s goal is to maintain a minimum of 99 percent accuracy once the time is set by the user. This functionality will be supported by ensuring that the processes are completed within their separated loops within the program to avoid disrupting the time’s flow of operations. The time is goes through a check to ensure that it stays within the limitations of actual time of sixty seconds, sixty minutes, and twenty-four hours in a day. The accuracy of the clock will be tested by changing the time 10 times and assessing how accurate the time stays for one minute each time.

Appendix A – Technical Issue and Risk Log

	Issues and Risk Log								
Issue or Risk	**Description**	**Project Impact**	**Action Plan/Resolution**	**Owner**	**Status**	**Importance**	**Date Entered**	**Date to Review**	**Date Resolved**
I/R	What is the issue or risk?	How will this impact scope, schedule, and cost?	How do you intend to deal with this issue?	Who manages this issue?	Where is this at?	Medium	*Date*	*Date*	*Date*
R	Not ramping up enough on the general knowledge required to program an FPGA board.	This would lead to abandoning the project if the training plan is not adhered to.	Complete the training plan.	Joeseph	The training plan to implement the clock's program has been completed.	High	*9-8-22*	*10-20-22*	*10-20-22*
R	If Go Lang can't replace C programming language.	Go Lang not having the capabilities of replacing C would be devastating to the project if that is one of the initial goals.	This has been placed in the out-of-scope list to be pulled in scope later if there is still time for a proof of concept at the end of the project.	Joeseph	Go Lang can be used in place of or in conjunction with C for a program.	High	*9-8-22*	*10-20-22*	11-1-22
R	Coding the FPGA components incorrectly.	This will enable the ability to complete the required feature of the project.	Complete training plan to become familiar with each component of the board.	Joeseph	The training plan for each component being utilized has been completed.	High	*9-8-22*	*10-20-22*	*10-20-22*
R	Not properly integrating fundamental FPGA programming concepts in the project.	This would lead to not having the project completed by the deadline.	Keep on schedule and plan out all tasks to ensure they are completed.	Joeseph	The training plan for each component being utilized has been completed.	High	*9-8-22*	*10-20-22*	*10-20-22*
R	Not adhering to the VHDL programming standards.	This would not meet the project completion requirement for this project.	Learn about VHDL programming standards and implement them into the project.	Joeseph	The training plan covers an introductory to this concept, so this will be completed.	High	*9-8-22*	*10-20-22*	*10-20-22*

Appendix B – References

Terasic, (2018), "DE10-Standard_User_manual"

Terasic, (2017), "DE10-Standard_Computer_ARM"

Sande, Joeseph, (2022), AlarmConfigFlowchart.png

Sande, Joeseph, (2022), AlarmStatusFlowchart.png

Sande, Joeseph, (2022), ClockStatusFlowchart.png

Sande, Joeseph, (2022), ClockConfigFlowchart.png

Sande, Joeseph, (2022), MainClockFlowchart.png

Appendix C – External Resources

GIT URL:	*The GIT URL (if applicable).*