CSE 379 Design Project

Robert DeBortoli

05/04/2015

Table of Contents

Section 1: The Introduction

1.1 Introduction

This document serves to describe in detail the design of a functional alarm clock. Functions that must be included are displaying the time, buzzing for the alarm, allowing the user to set the time, set the alarm, turn off the alarm, and snooze. The following components that should be used to complete these functionalities are: An ARM microprocessor, UART, PrimeCell RTC module, PrimeCell GPIO module, 4 momentary push buttons, 3 LEDs, a buzzer, as well as a keypad. Additional system details include: 1 MB ROM, 2MB RAM, and partial decoding must be used. The AMBA system is also utilized.

1.2 Use case overview:

Basic operation of the clock will go as such:

- To start, power on the system and then set the time by punching in 4 consecutive digits on the keypad and pressing the set-time momentary push button.
- Set alarm proceeds in a similar manner.
- To snooze or disable the alarm simply press the corresponding momentary push buttons. Snoozing the alarm momentarily disables it for 10 minutes.
- All digits that are not pressed into the keypad will be assumed to be 0. For example, if the buttons 1, 0, and then “set time” are pressed, then the time that will be taken in is 10:00.

1.3 Conventions:

Conventions to be used for this document include:

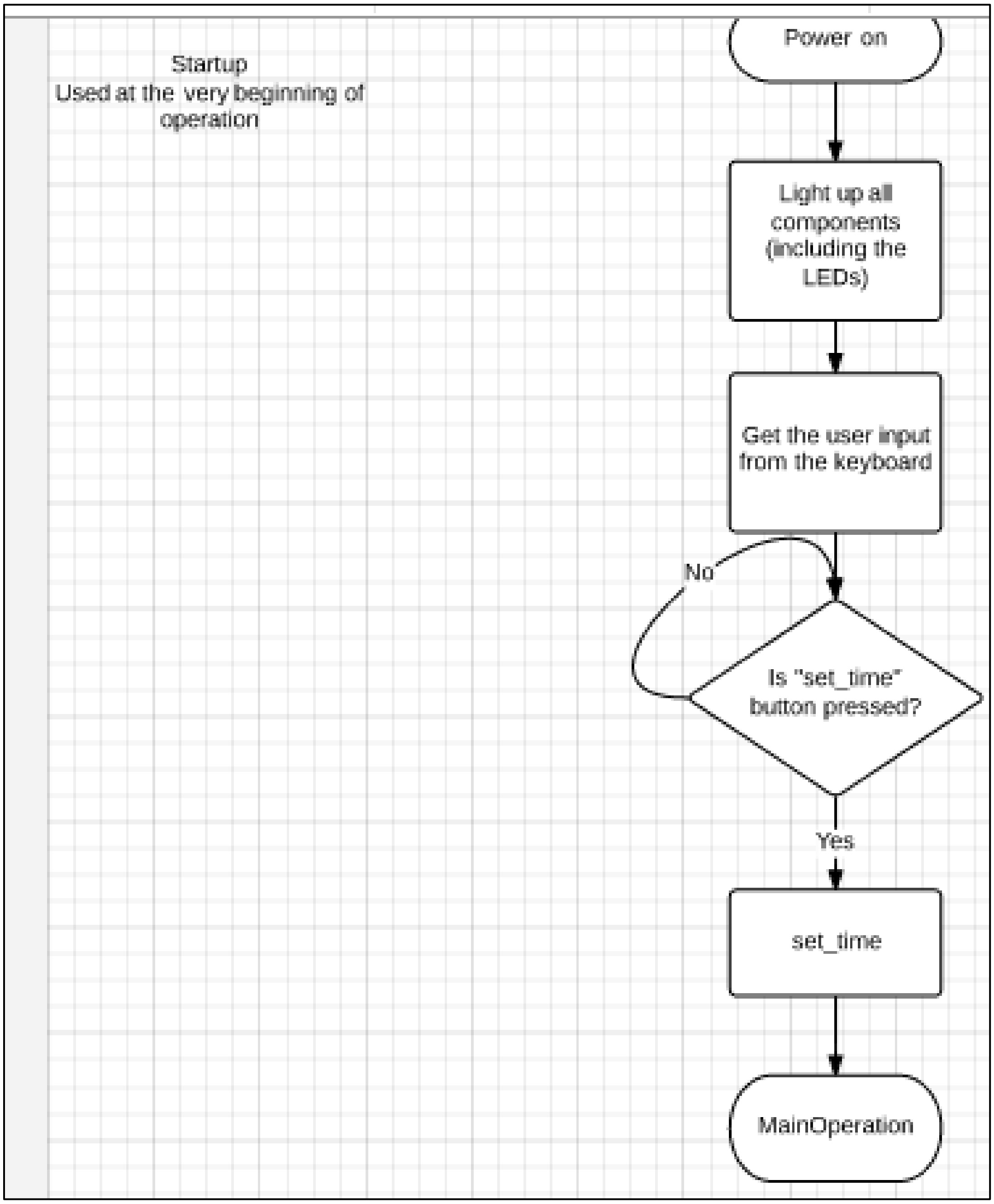
- P0-P3 are the 4 momentary push buttons, in the following order: set_time, set_alarm, snooze, and diable_alarm.
- LED1 and LED2 are the colons for the time. They are always lit. LED3 is the AM/PM LED. It is lit when the time is in the PM.
- TR is the time register in memory (stored in clock cycles)
- AR is the alarm register in memory (stored in clock cycles)

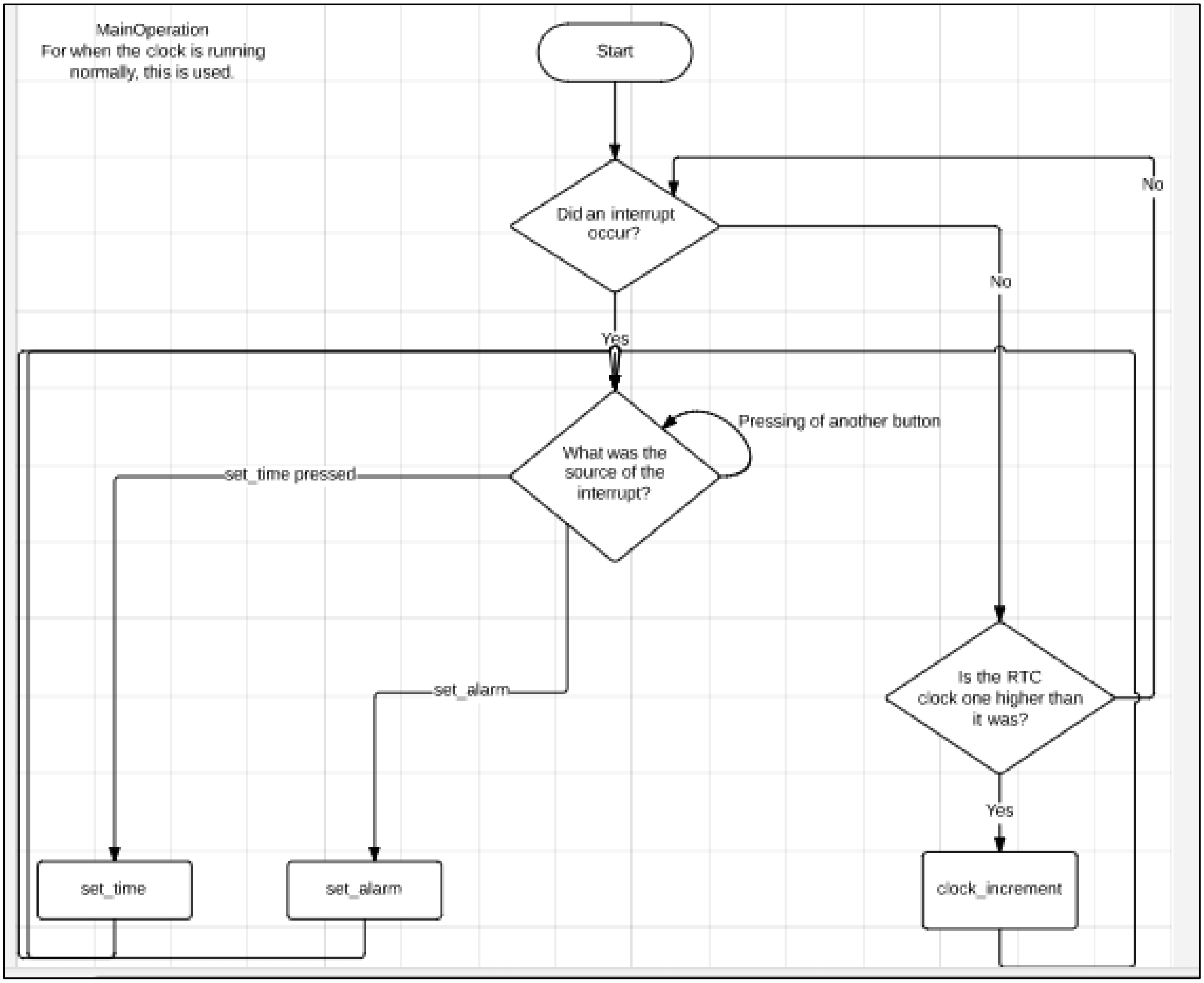
1.4 Parts list:

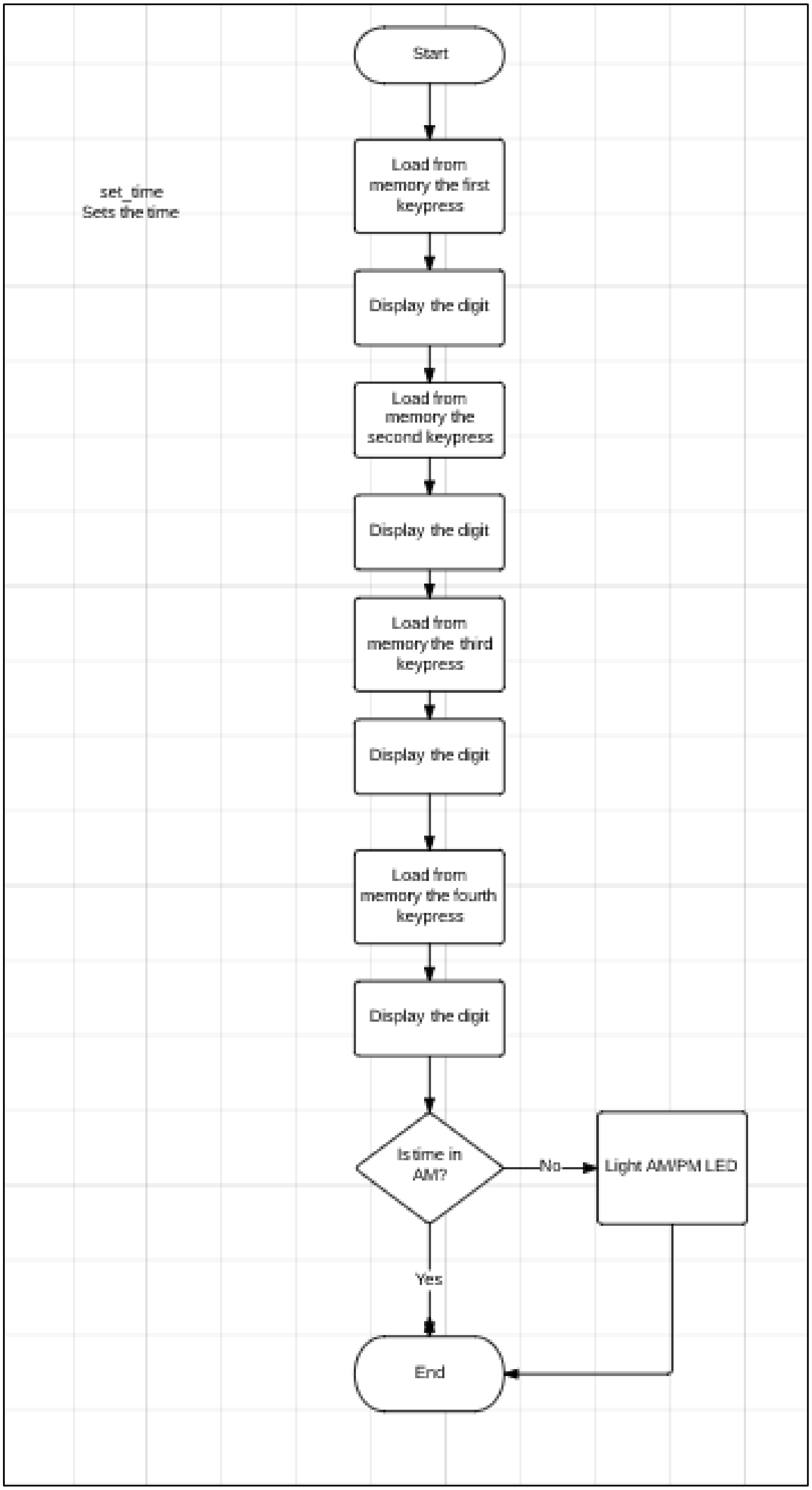
Name	Manufacturer	Part Number	Quantity
ARM7TDMI Microprocessor	ARM		1
7-Sement display	China Young Sun LE Technology Co.	YSD-160AR4B-8	4
LED	China Young Sun LE Technology Co.	YSL-R531R3D-D2	3
Push button	Sparkfun	COM-11966	4
Buzzer	CUI Inc.	CEM-1203(42)	1
Keypad	Sparkfun	COM-08653	1
Parallel to Serial Converter	Micrel	SY10E446	1

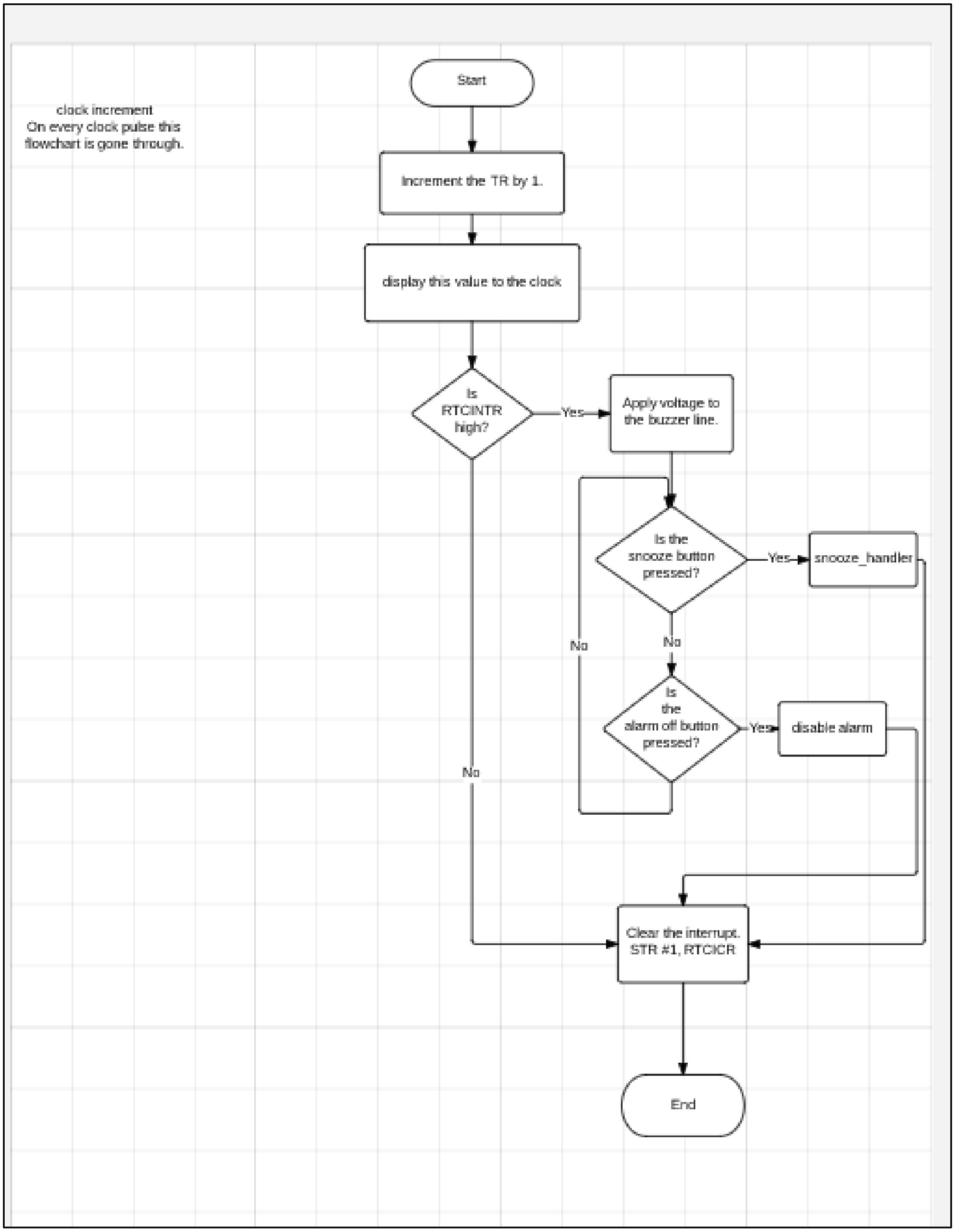
Section 2: The Flowcharts

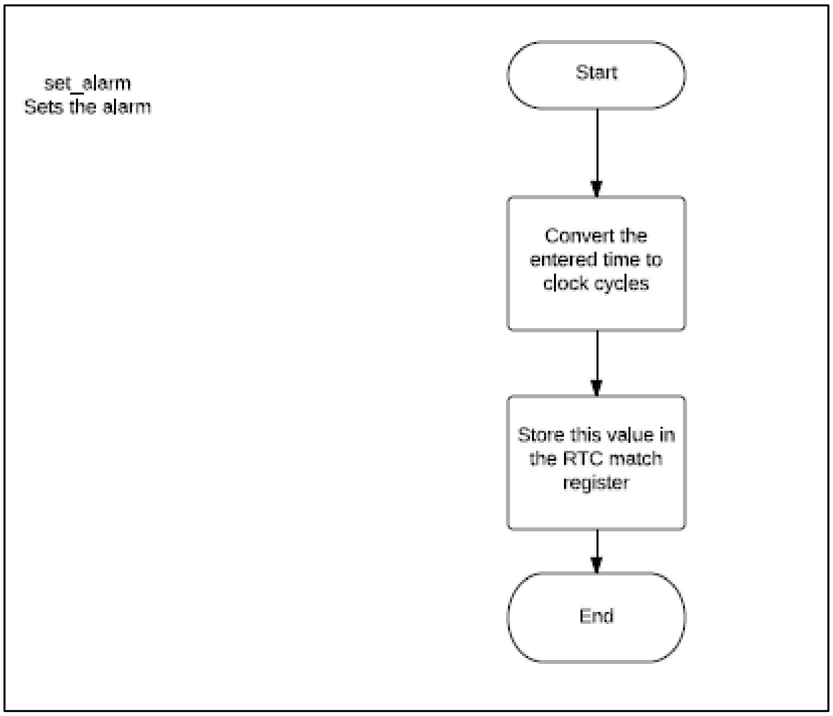
To demonstrate the flow of operation by the user, for the purpose of programming this system, the following flowcharts were designed to organize this information. These are very general as the scope of the project calls for this. For a more detailed approach to programming this system please review the entire document, which contains pertinent and specific information on this topic.

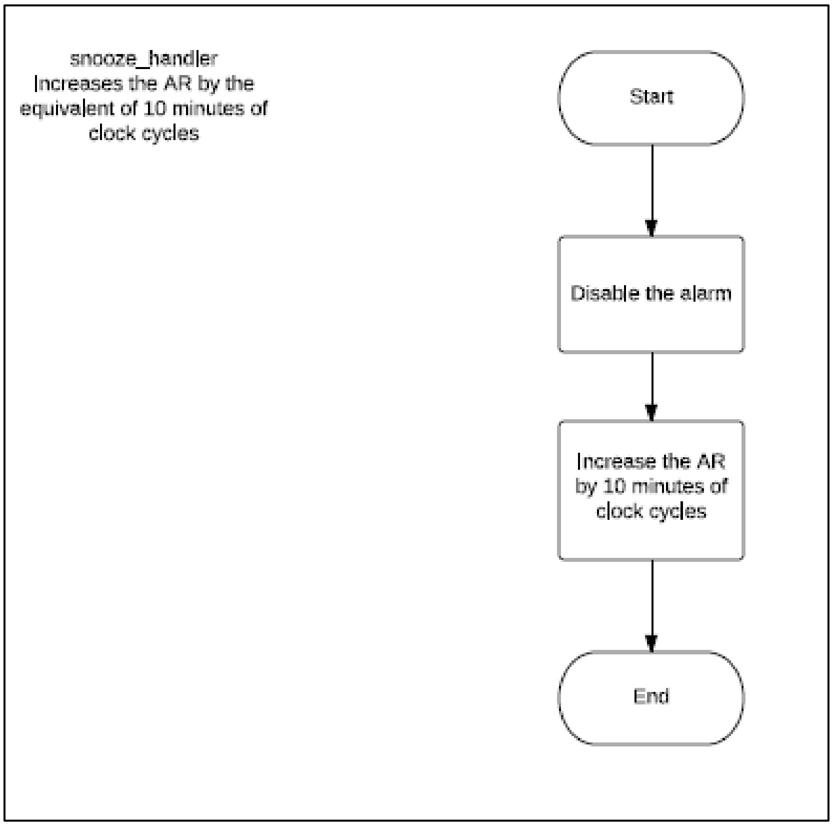

Startup
Used at the very beginning of operation
Power on
Light up all components (including the LEDs)
Get the user input from the keyboard
No
Is "set_time" button pressed?
Yes
set_time
MainOperation


MainOperation
For when the clock is running normally, this is used.
Start
Did an interrupt occur?
No
No
Yes
What was the source of the interrupt?
Pressing of another button
set_time pressed
set_alarm
Is the RTC clock one higher than it was?
Yes
set_time
set_alarm
clock_increment


set_time
Sets the time
Start
Load from memory the first keypress
Display the digit
Load from memory the second keypress
Display the digit
Load from memory the third keypress
Display the digit
Load from memory the fourth keypress
Display the digit
Is time in AM?
No
Light AM/PM LED
Yes
End


clock increment
On every clock pulse this flowchart is gone through.
Start
Increment the TR by 1.
display this value to the clock
Is RTCINTR high?
Yes
Apply voltage to the buzzer line.
Is the snooze button pressed?
Yes
snooze_handler
No
No
Is the alarm off button pressed?
Yes
disable alarm
No
Clear the interrupt. STR #1, RTCICR
End


set_alarm
Sets the alarm
Start
Convert the entered time to clock cycles
Store this value in the RTC match register
End


snooze_handler
Increases the AR by the equivalent of 10 minutes of clock cycles
Start
Disable the alarm
Increase the AR by 10 minutes of clock cycles
End

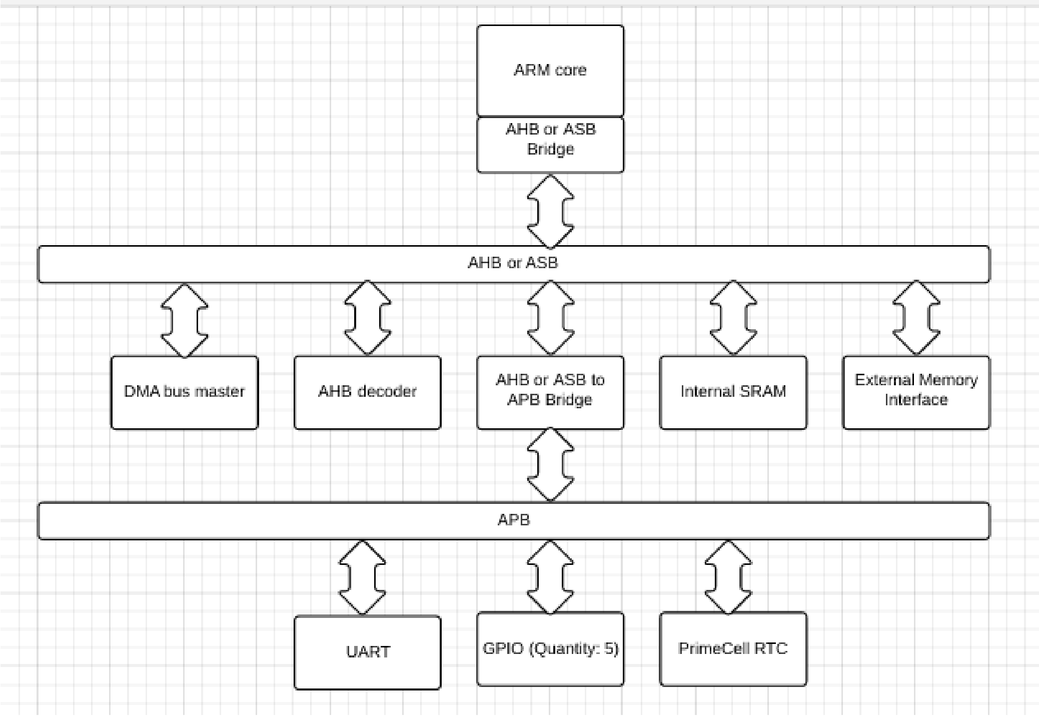
Sections 3: The General Hardware Diagrams

General hardware diagrams:

This sections details overview system organization in a variety of forms and areas including: An AMBA Overview, Bus Interface Diagrams, Memory System Diagram, Memory Map Diagram, a General and Specific Decoder Logic Diagram, and finally a UART to keypad diagram.

3.1 AMBA overview

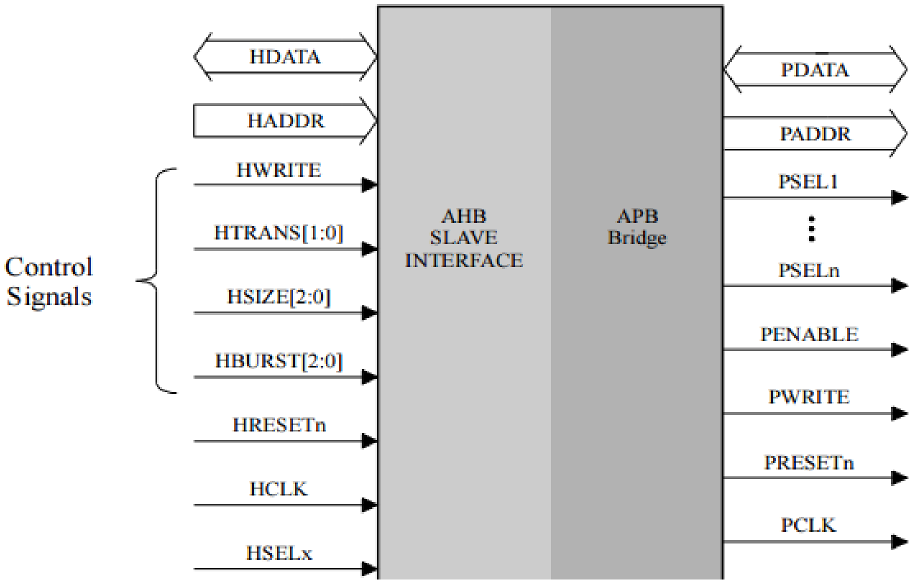
The following is an overview of how this system utilizes the AMBA architecture. In this system there are 5 PrimeCell GPIO modules however for cleanliness of the diagram, this is simply notated with (Quantity: 5) instead of drawing each individual module.



3.2 Bus Interface Diagram

The following is a table and accompanying diagram detailing how the AHB Slave interface and APB Bridge interact by describing the signals associated with each. After that ensues a discussion (where appropriate) on the relevance of these signals to the system is detailed. The AHB is the High-Performance Bus which is used to interface the processor with modules such as memory and decoders, which can handle the high speeds (thus the name). The APB is the low power Peripheral Bus, which interfaces with components such as the GPIOs and the RTC.

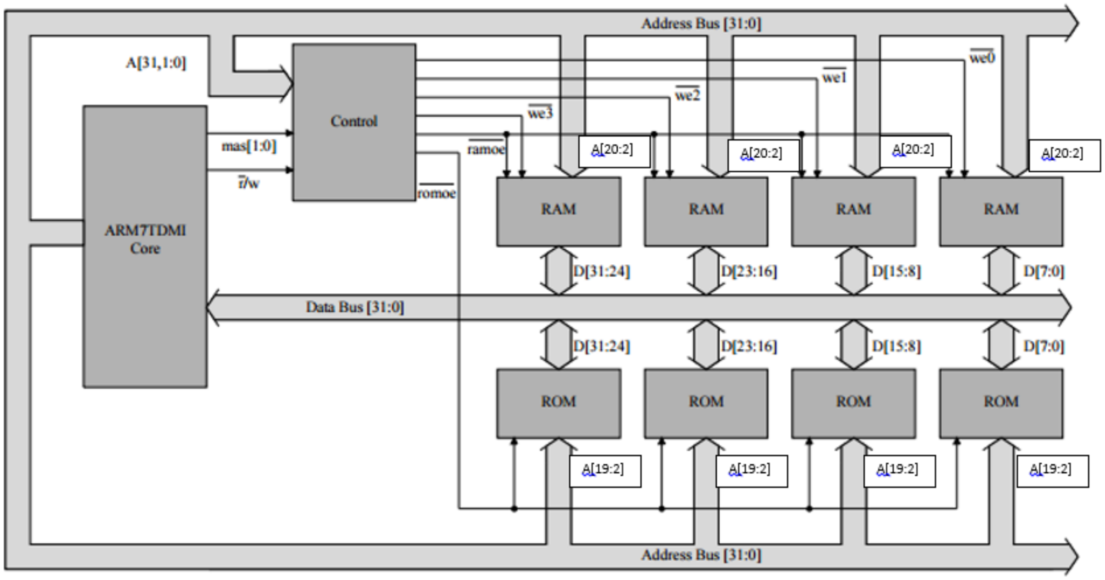
Name	Description
HSELx	Selects the device on the AHB
PSELx	Selects the device on the APB. This is done by decoding HADDR.
HCLK, PCLK	Simple clocks, They are synchronized so that the AHB and APB are synchronized on their transactions.
HRESETn	An active low signal, it resets the AHB. PRESETn is also tied to this.
HWRITE	Used for transferring. 0=>read//1=>write. PWRITE works in the same way
HTRANS	Selects the transfer type from the following options: Busy, Sequential, Non-sequential, and Idle.
HBURST	Details whether the transfer is part of a burst transfer or not.
HSIZE	Details the size of the transfer (byte, halfword, word).



Additional notes relevant to this specific system are now discussed. PSEL1…PSELn can be utilized for the following components: GPIO1-4, the UART, and the RTC. These PSEL signals determine which peripheral is being utilized. PDATA is used to transfer data from these peripherals to memory. Communication from AHB level modules (such as the memory) to the peripherals (such as the GPIOs) s done via HDATA.

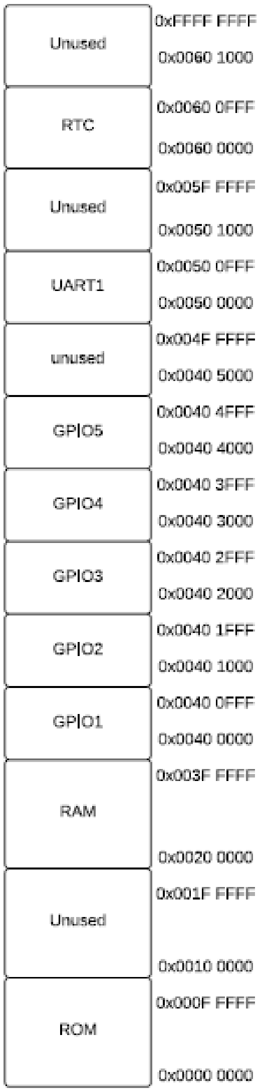
3.3 Memory System

Because RAM requires 18 bits to be addressed (2^{18} = 262,144) and it is byte addressed, line [18:2] are used to access RAM. Because ROM requires 17 bits to be addressed (2^{17}=131,072) and it is also byte addressed, lines [17:2] are used to access ROM. Because the data is accessed byte by byte and each chip of RAM has the same amount of memory as the other RAM chips and each ROM chip has the same amount of memory as the other ROM chips, the data bus breaks up the addressing in groups of 8 ([31:24], [23:16], [15:8], [7:0]).



3.4 Memory Map

This memory map serves to show the memory for the alarm clock and how it is divided. This is also useful when formulating decoder logic.

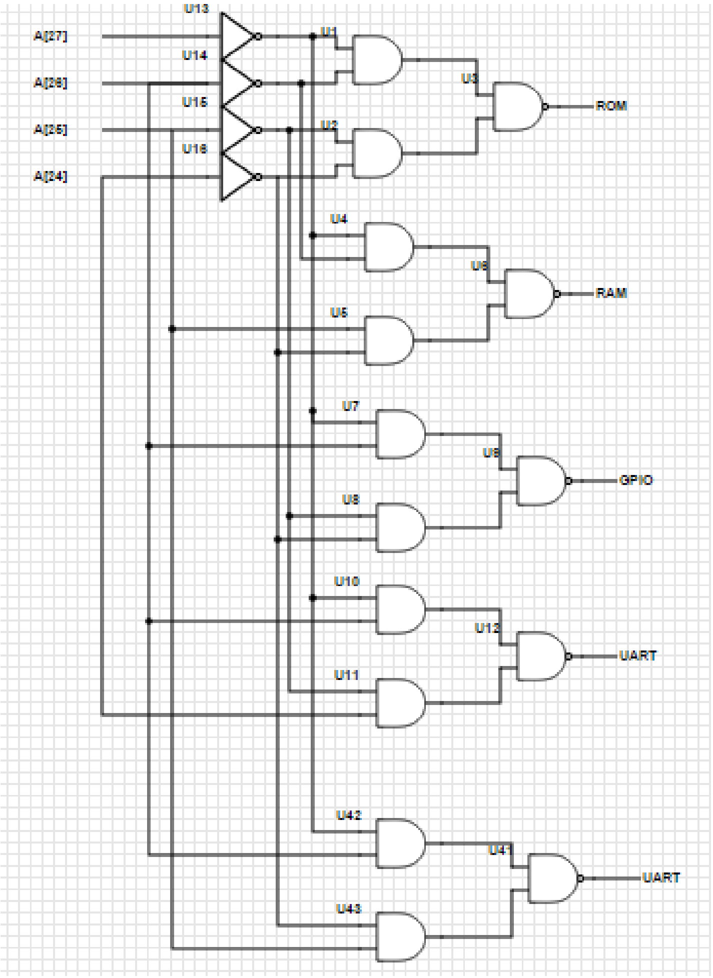


Unused	0xFFFF FFFF	0x0060 1000
RTC	0x0060 0FFF	0x0060 0000
Unused	0x005F FFFF	0x0050 1000
UART1	0x0050 0FFF	0x0050 0000
unused	0x004F FFFF	0x0040 5000
GPIO5	0x0040 4FFF	0x0040 4000
GPIO4	0x0040 3FFF	0x0040 3000
GPIO3	0x0040 2FFF	0x0040 2000
GPIO2	0x0040 1FFF	0x0040 1000
GPIO1	0x0040 0FFF	0x0040 0000
RAM	0x003F FFFF	0x0020 0000
Unused	0x001F FFFF	0x0010 0000
ROM	0x000F FFFF	0x0000 0000

3.5 General Decoder Logic Diagram

Below is a schematic for the decoder logic detailing which part of the system is accessed depending on the address into the decoder. The four lines monitored are A[27:24]. The following truth table shows this addressing system bit by bit.

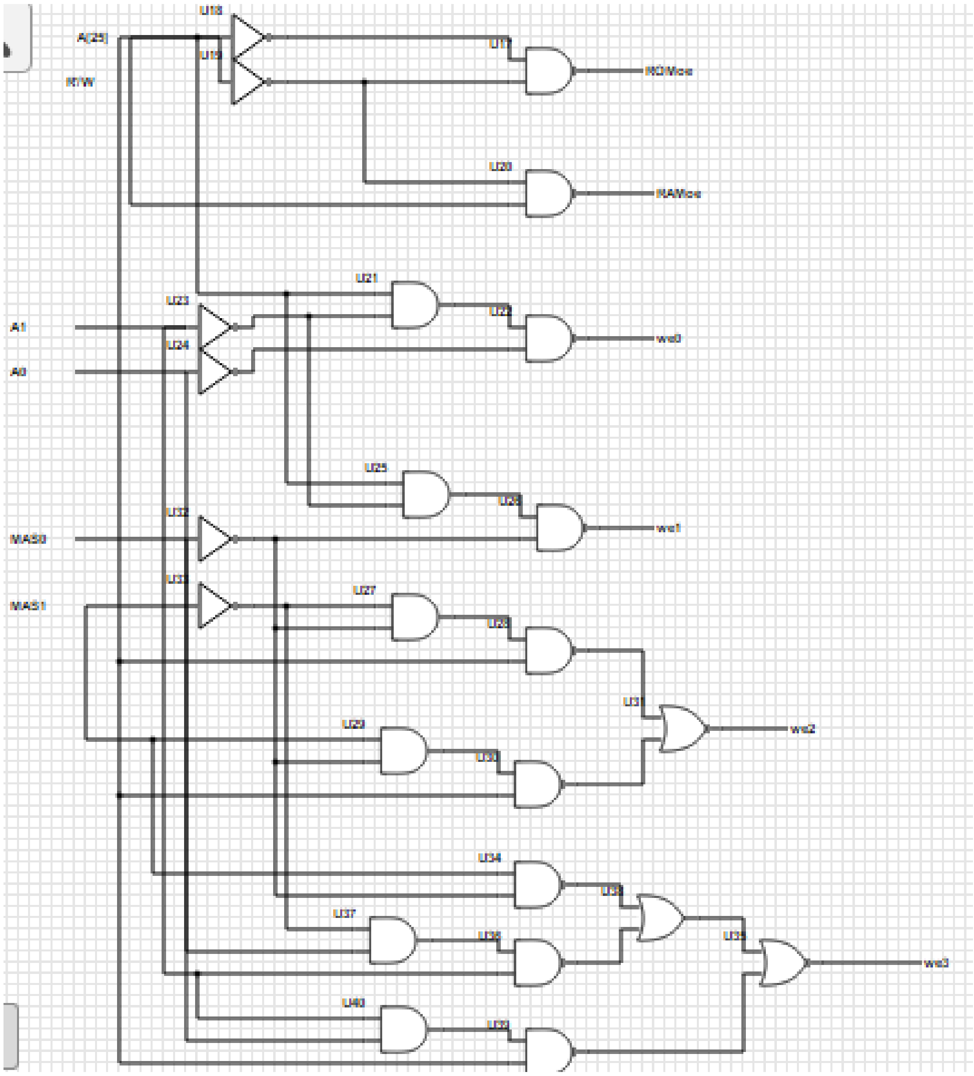
A[27]	A[26]	A[25]	A[24]	Component of system
0	0	0	0	ROM
0	0	1	0	RAM
0	1	0	0	GPIO
0	1	0	1	UART
0	1	1	0	RTC



3.6 Specific Decoder Design

The next natural area to examine is a decoder for the RAM/ROM operations. The next decoder to look at is the one mapping address lines to ROMoe, RAMoe, and lines such as we0-we3. The truth table is shown below and a logical representation of this same information follows.

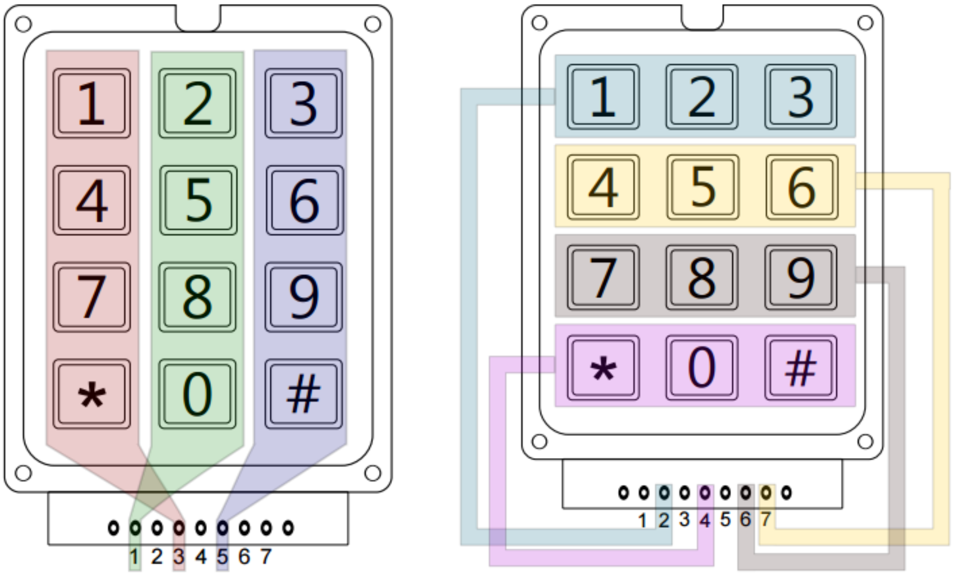
Operation	A25	MAS1	MAS0	A1	A0	R’/W	ROMoe	RAMoe	we0	we1	we2	we3
ROM Read	0	-	-	-	-	0	0	1	1	1	1	1
RAM Read	1	-	-	-	-	0	1	0	1	1	1	1
RAM Write (Word)	1	1	0	0	0	1	1	1	0	0	0	0
RAM Write (Halfword)	1	0	1	0	0	1	1	1	0	0	1	1
RAM Write (Halfword)	1	0	1	1	0	1	1	1	1	1	0	0
RAM Write (Byte)	1	0	0	0	0	1	1	1	0	1	1	1
RAM Write (Byte)	1	0	0	0	1	1	1	1	1	0	1	1
RAM Write (Byte)	1	0	0	1	0	1	1	1	1	1	0	1
RAM Write (Byte)	1	0	0	1	1	1	1	1	1	1	1	0


U18
A[25]
U19
R/W
U17
ROMoe
U20
RAMoe
U21
U23
U22
A1
U24
we0
A0
U25
U26
U32
MAS0
we1
U33
MAS1
U27
U28
U31
we2
U29
U30
U34
U36
U37
U38
U35
we3
U40
U39

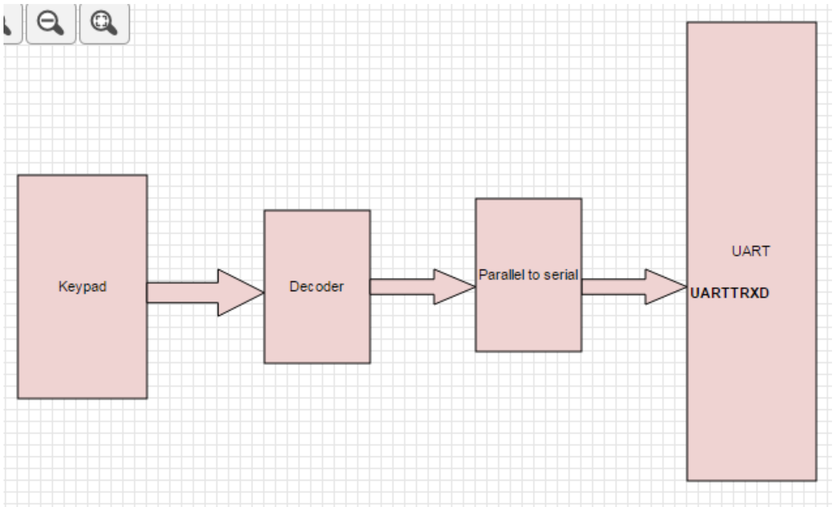
3.7 UART to Keypad Design

The project specification requires the use of a UART to take information from a keypad and use it accordingly. This design takes a keypad, uses a decoder to decode the information from this keypad, and uses a parallel to serial converter to send this data to the UART. The following includes a general description of the system, diagram of the overall system, a truth table for the decoder, and a discussion on considerations for the connections of these hardware components. Wire-to-wire diagrams were deemed out of the scope of this project after a discussion with Rebecca.

In general the keypad utilizes 7 pins (shown below at the bottom of the component) and the scheme shown below to produce values. These values are put into a decoder designed by the engineer, which are then serialized using a parallel to serial converter, which is then utilized by the UART. An example is useful in this situation. Take for example the user pressing the "1" button. These sets pin 3 high because it is in the first column and pin 2 high because it is in the first row. Therefore, we as the engineer know that a one was pressed because pins 2 and 3 only are high.



The overall system. As one can see the input from the keypad goes through decoder logic and a "serializer" to arrive at the UART, which receives the data into its UARTTRXD register. This data is then available to the rest of the system.



The truth table for the decoder:

Output	***P1***	***P2***	***P3***	***P4***	***P5***	***P6***	***P7***
0	*1*			*1*			
1		*1*	*1*				
2	*1*	*1*					
3		*1*			*1*		
4			*1*				*1*
5	*1*						*1*
6					*1*		*1*
7			*1*			*1*	
8	*1*					*1*	
9					*1*	*1*	

Section 4: The Component Specific Diagrams

Component-specific overviews:

The next logical step to detail is the function and purpose of each specific component in this system. After this schematics will be shown to physically demonstrate the component's place in the system.

4.1 PrimeCell UART Overview:

The PrimeCell Universal Asynchronous Receiver Transmitter (UART) is used in this system for serial communication. Serial communication in this context means transmitting information one bit at a time. The UART connects to the APB. This specific UART also features a FIFO buffer for receiving and transmitting data. A brief description of the component follows which includes initialization as well as receiving/transmitting data and interrupt support.

Initialization is done first by setting the baud rate (or the rate at which bits and transferred). The following formula is used:

$$Baud\ rate = \frac{F_{UARTCLK}}{16 * Baud\ rate\ divisor}$$

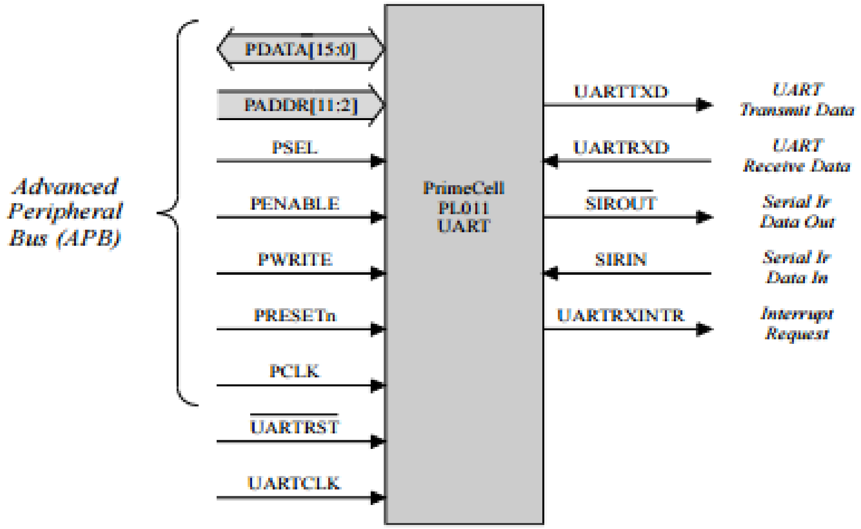
The baud rate divisor is comprised of the value in two registers: The 16 bit Integer Baud Rate Divisor Register (UARTIBRD) and the 6 bit Fractional Baud Rate Divisor Register (UARTFBRD). The last initialization register detailed is the 30 bit UART Line Control Register. The following table describes its composition:

Name	Bits	Description
Line Control Register	[29:22]	Selects parity, word length, FIFO enable, number of stop bits, among other properties
Integer Baud Rate Divisor	[21:6]	see above
Fractional Baud Rate Divisor	[5:0]	see above

Progressing to receiving and transmitting data, this is done through the UART Data Register (UARTDR). When reading data 4 status bits and 8 data bits are returned. When writing data 8 data bits are transmitted.

Finally interrupts are implemented using the UART Interrupt Mask Set/Clear Register (UARTIMSC). This can be read to determine if an interrupt has occurred. Clearing this interrupt is done using the Interrupt Clear Register (UARTICR).

The PrimeCell UART and the pins described above are demonstrated below.

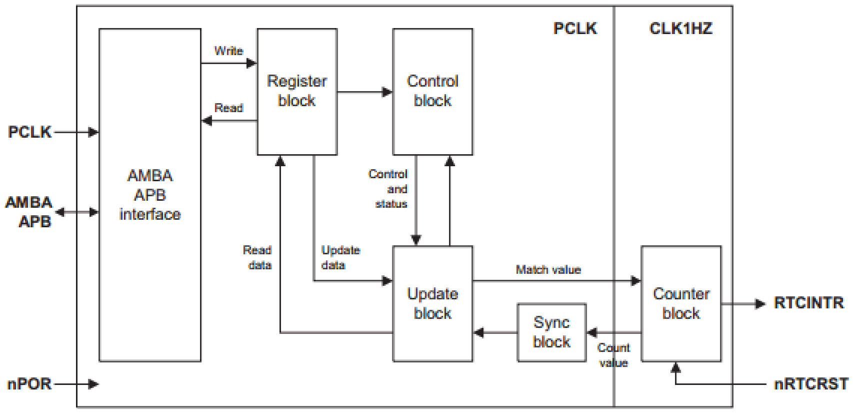


4.2 PrimeCell RTC Overview:

The RTC component is a simple up counter using a 1Hz clock. It is connected to the APB of the AMBA interface. A match register is used to generate an interrupt. The following lists important registers and their roles in the system:

- **RTCDR** -- Data Register – Read to give the count.
- **RTCMR** -- Match Register – When the count equals this register, an interrupt is generated.
- **RTCINTR** – Interrupt Register – When the data register and match register are equal, this register is asserted high.
- **RTCICR** – Interrupt Clear Register – Writing a 1 to bit position 0 clears the corresponding interrupt.

The following is a functional diagram of the PrimeCell RTC as well as a discussion of each block and finally a summary of all applicable registers for the RTC. The functional diagram and register summary are directly from the ARM documentation on this module. Because the descriptions are quite short in the documentation for these items, the descriptions follow very closely to the descriptions in the documentation.



The sections of this diagram include:

- ***Register block**: stores the data to be transmitted across the APB Interface.*
- ***Control block:** Controls the updating of the RTC value as well as the generation of the interrupt associated with the module.*
- ***Update block:** is used to determine if an interrupt has occurred (by using the match register) as well as updating the value of the RTC when two rising edges of the PCLK has occurred.*
- ***Synchronization block:** synchronizes the two clocks involved (PCLK and CLK1Hz). By synchronizing these clocks, it also synchronizes determination of the equality of the match register and the RTC value.*

- ***Counter block:*** *Simply an up counter running at 1Hz. It is 32 bits and ranges from 0x00000000 to 0xFFFFFFFF. When it reaches 0xFFFFFFFF it wraps around to 0x00000000.*
- ***Test register and logic:*** *out of this project scope. Used at the manufacturing and non-consumer level to test the different functional areas of this system.*

Finally in this section, a largely comprehensive list of the RTC registers:

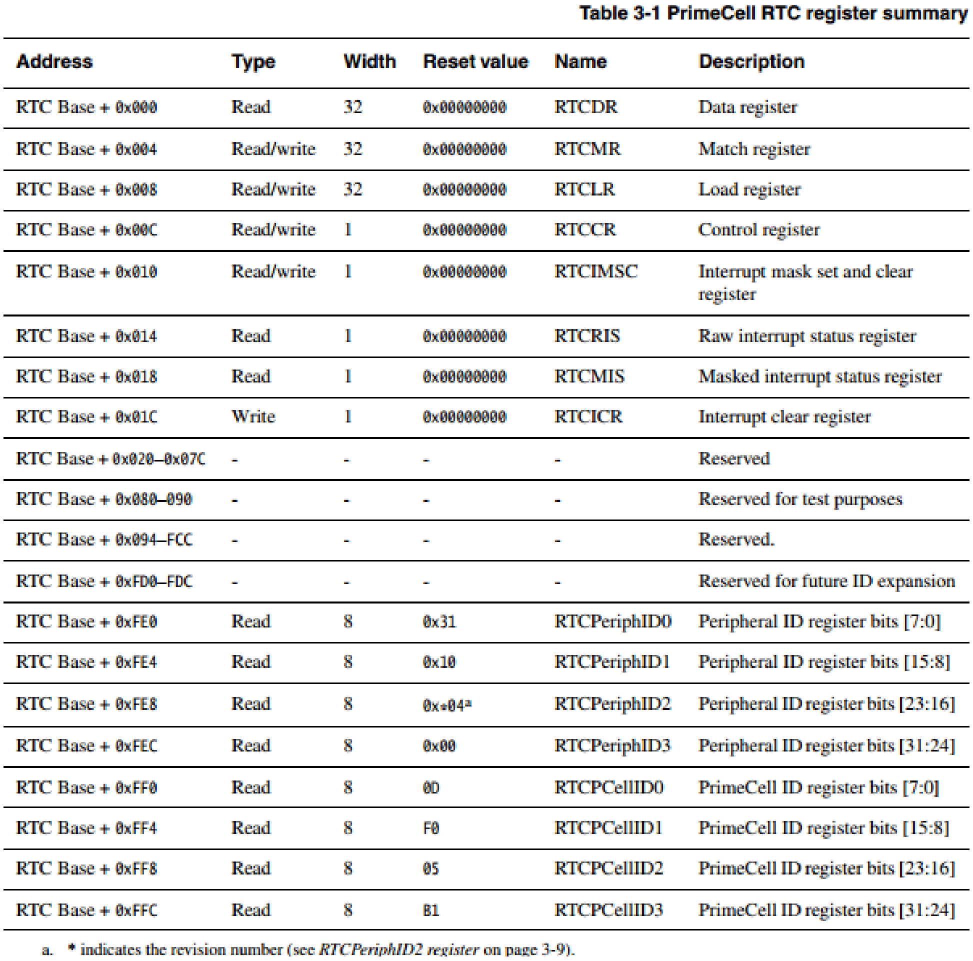


Table 3-1 PrimeCell RTC register summary

Address	Type	Width	Reset value	Name	Description
RTC Base + 0x000	Read	32	0x00000000	RTCDR	Data register
RTC Base + 0x004	Read/write	32	0x00000000	RTCMR	Match register
RTC Base + 0x008	Read/write	32	0x00000000	RTCLR	Load register
RTC Base + 0x00C	Read/write	1	0x00000000	RTCCR	Control register
RTC Base + 0x010	Read/write	1	0x00000000	RTCIMSC	Interrupt mask set and clear register
RTC Base + 0x014	Read	1	0x00000000	RTCRIS	Raw interrupt status register
RTC Base + 0x018	Read	1	0x00000000	RTCMIS	Masked interrupt status register
RTC Base + 0x01C	Write	1	0x00000000	RTCICR	Interrupt clear register
RTC Base + 0x020–0x07C	-	-	-	-	Reserved
RTC Base + 0x080–090	-	-	-	-	Reserved for test purposes
RTC Base + 0x094–FCC	-	-	-	-	Reserved.
RTC Base + 0xFD0–FDC	-	-	-	-	Reserved for future ID expansion
RTC Base + 0xFE0	Read	8	0x31	RTCPeriphID0	Peripheral ID register bits [7:0]
RTC Base + 0xFE4	Read	8	0x10	RTCPeriphID1	Peripheral ID register bits [15:8]
RTC Base + 0xFE8	Read	8	0x*04[a]	RTCPeriphID2	Peripheral ID register bits [23:16]
RTC Base + 0xFEC	Read	8	0x00	RTCPeriphID3	Peripheral ID register bits [31:24]
RTC Base + 0xFF0	Read	8	0D	RTCPCellID0	PrimeCell ID register bits [7:0]
RTC Base + 0xFF4	Read	8	F0	RTCPCellID1	PrimeCell ID register bits [15:8]
RTC Base + 0xFF8	Read	8	05	RTCPCellID2	PrimeCell ID register bits [23:16]
RTC Base + 0xFFC	Read	8	B1	RTCPCellID3	PrimeCell ID register bits [31:24]

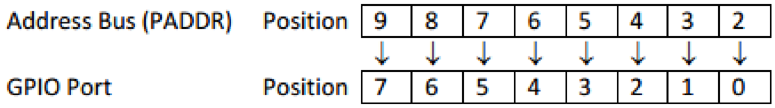
[a] * indicates the revision number (see *RTCPeriphID2 register* on page 3-9).

4.3 PrimeCell GPIO Overview:

The GPIO will be covered in 3 main sections: first initialization of the component, then reading to and writing from the GPIO module, and finally interrupt operation on the GPIO.

Initialization is completed mainly by utilizing 2 registers: the Mode Control Register and the Data Direction Register. The Mode Control Register configures the ports for software (value =0) or hardware (value = 1) control. The value of the Data Direction Register determines whether each of the software controlled ports and inputs (value =0) or outputs (value = 1).

Reading and writing (or more generally transmitting data) is done with an 8-bit data register. It also has a masking feature for data which is demonstrated by the image shown below:



Address Bus (PADDR)	Position	9	8	7	6	5	4	3	2
		↓	↓	↓	↓	↓	↓	↓	↓
GPIO Port	Position	7	6	5	4	3	2	1	0

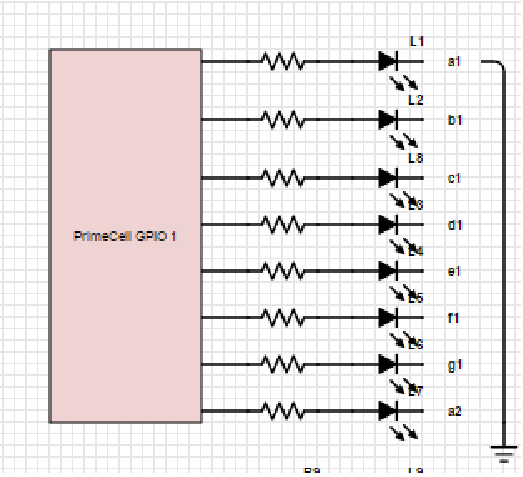
This masking allows for data to be changed (value = 1) or not changed (value = 0).

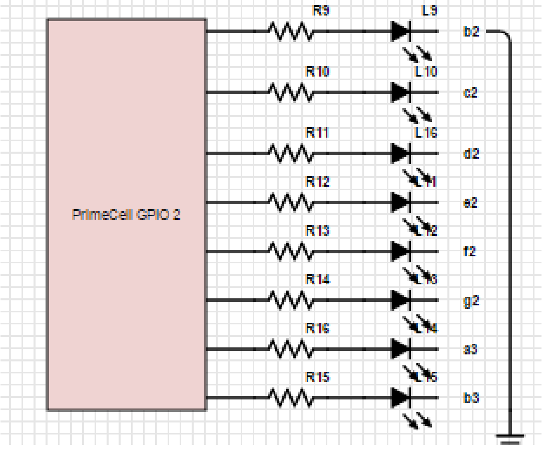
Finally, the GPIO can generate a single interrupt (GPIOINTR) when a 1 or more of the GPIO lines cause an interrupt. This is done through the utilization of 5 registers outlined below:

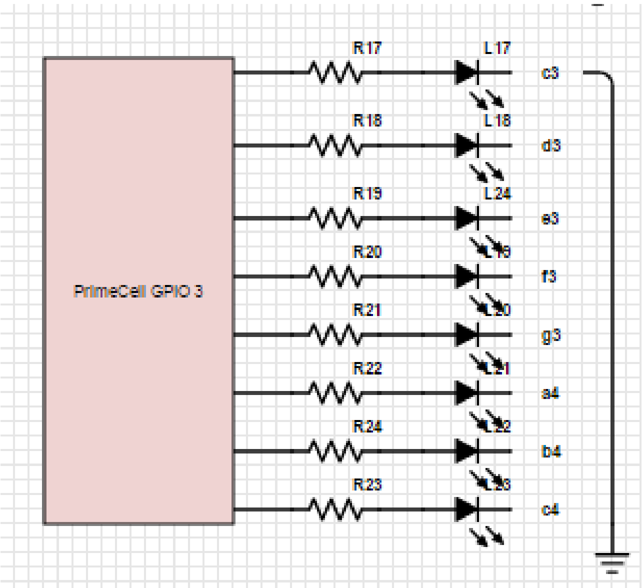
Name	Description
Interrupt Sense Register (GPIOIS)	Configures a port for detecting an interrupt on either level or transition clock pulses. 0=>edge//1=>level
Interrupt Event Register (GPIOEV)	Based on GPIOIS this configures the interrupt to occur on rising or falling edges. 0=>high(rising)//1=>low(falling)
Interrupt Mask Register (GPIOIE)	Allows a port to trigger and interrupt. 0=>disable//1=>enable
Interrupt both edges register(GPIOIBE)	This register allows for the interrupts to be dual edge triggered. 0=>controlled by GPIOIE//1=>dual-edge triggered
Interrupt clear register (GPIOICE)	The clears the interrupt for a port where edge detection is enabled. 0=>nothing//1=>clear

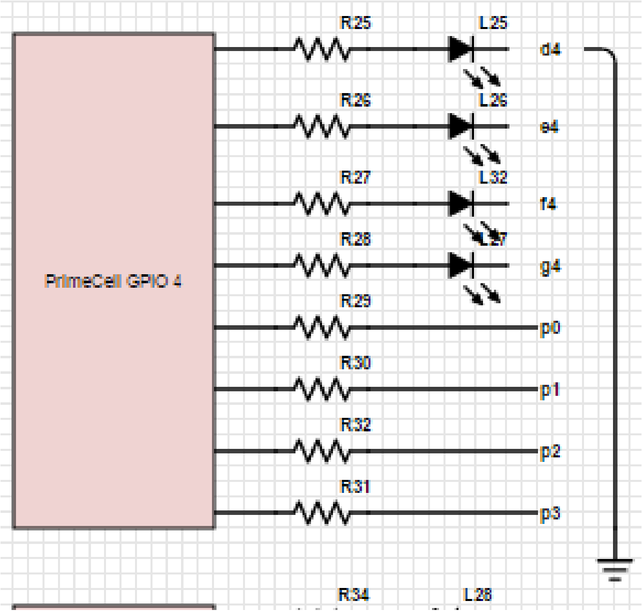
GPIO connections

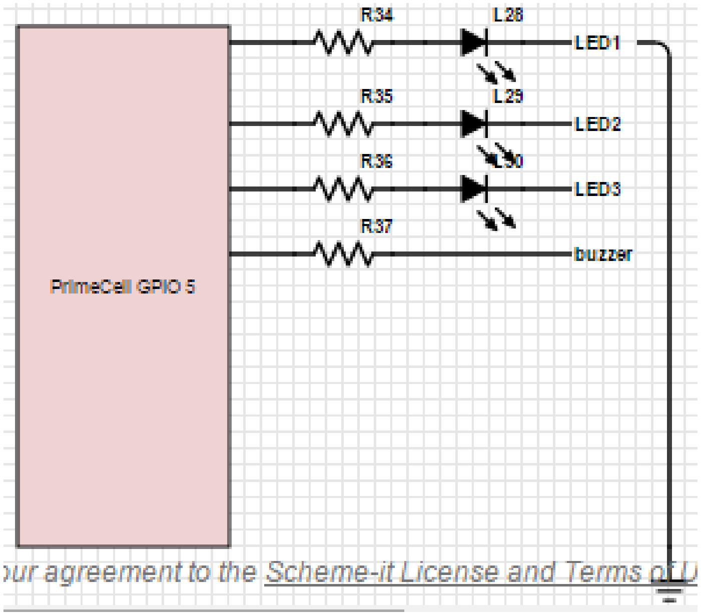
The following are 5 diagrams showing in detail the connections from each port of each GPIO to each external component. The circuit design software used does not explicitly show each port being grounded, however it may be assumed that each connection is grounded (see the bus at the far right of every image. GPIO modules 1, 2, and 3 all show the connection of each port to a resistor, LED, and to ground. Each LED may be assumed to be a segment in a 7 segment display and they are labelled as such (for example the first display segment a is labelled a1). The first 5 ports of GPIO4 are used for the same purpose as 1, 2, and 3. The last 4 ports of GPIO4 are used for push button support. Each push button is labelled as px where x is the number of the push button. Please refer to the "Conventions" section to see what this numbering means. GPIO5 contains the LEDs1-3 as well as the buzzer. Please refer to the conventions section to view refer to information regarding this numbering scheme.






PrimeCell GPIO 3
R17
L17
c3
R18
L18
d3
R19
L24
e3
R20
L19
f3
R21
L20
g3
R22
L21
a4
R24
L22
b4
R23
L23
c4


PrimeCell GPIO 4
R25
L25
d4
R26
L26
e4
R27
L32
f4
R28
L27
g4
R29
p0
R30
p1
R32
p2
R31
p3
R34
L28


R34
L28
LED1
R35
L29
LED2
R36
L30
LED3
R37
buzzer
PrimeCell GPIO 5
our agreement to the Scheme-it License and Terms of U

Section 5: The References

References:

[I] Kris Schindler, Introduction to Microprocessor Based Systems Using the ARM Processor, Pearson, 2012

[II] UM10120 - Volume 1: LPC213x User Manual, Rev. 01, Philips, June 24, 2005

[III] Andrew N. Sloss, Dominic Symes, and Chris Wright, ARM System Developer's Guide - Designing and Optimizing System Software, Morgan Kaufmann, 2004

[IV] Steve Furber, ARM System-on-chip Architecture, 2nd Edition, Addison-Wesley, 2000 User's Manual KS32C50100 32-bit RISC Micro Controller Embedded Network Controller, Samsung Electronics, March 1999

[V] http://en.wikipedia.org/wiki/Microprocessor

[VI] Lectures given by Kris Schindler privately available at: http://www.cse.buffalo.edu/~kds/cse-379/lectures.html

[VII] Keypad documentation available at: https://www.sparkfun.com/products/8653

[VIII] General component documentation available privately through the course website at: http://www.cse.buffalo.edu/~kds/cse-379/documentation.html