THOMPSON RIVERS UNIVERSITY

Digital Systems Design (Fall 2021) - CENG 3010

# Controller for Remote Car Starter

Nazmul Hoque (T00624483)

Samuel Elgert (T00609291)

Kieren O’Neil (T00238338)

2021-11-04

## Table of Contents

## List of Figures

## List of Tables

## 1. Introduction

Canadians rely heavily on cars for transport and drive large distances with them. They own over the OECD average number of cars per capita and drive them further than the OECD average distance per car per year [1]. Combining this with the fact that Canadian winters are very harsh with some urban areas such as Edmonton which have multiple months with average temperatures below -10 ℃ [2]. Due to climate change, Canada will experience more extreme weather events including winter storms [3].

These factors mean remote car starters are a very important piece of technology for Canada and Canadians. Remote starters allow for cars to be easily and safely turned on to warm up while the driver and passengers can remain comfortably in a building rather than inside the cold car. These devices also come equipped with the ability to lock and unlock the doors and trunk of a car, start and stop the engine, and turn the heater on and off; all of which are extraordinarily useful features.

The purpose of this design project was to design a remote car starter and implement it on a Xilinx Basys 3 FPGA Board for testing. The goal of this design was to have a remote car starter to make it convenient for users to get into a car that has a comfortable inside temperature and a warmed-up engine. The final solution developed for this project implemented this functionality. This improved the functionality of a standard remote access fob for a car by adding more control to it creating a final device that would be more useful specifically in the winter months. It enabled the user to control the doors and trunk, engine, and heater of a car remotely.

## 2. Design Problem

### 2.1. Problem Definition

Design and create an automatic remote car starter whose operations are data-driven. The car starter allows the user to turn on and off the engine, open and close the trunk door and turn on and off the heater as needed.

### 2.2. Design Requirements

#### 2.2.1. Functions

To make our product met the requirements given, we developed the following functions for our remote car starter:

- Must lock and unlock the car doors
- Must  the car trunk
- Must start and turn off the engine of the car
- Must start and turn off the heater of the car
- Must indicate the status of the trunk, doors, engine, and heater of the car

#### 2.2.2. Objectives

We developed objectives that would guide us throughout our project.

Following are the objectives of the remote car starter:

- Efficiency: The remote control must consume minimal fuel by automatically turning off the engine when the car is idle for 2 minutes.
- Reliability: The controller must work 24/7, 365 days without any issues.
- Security: The controller must lock the car automatically if the user did not enter the car within 30 seconds.
- Environmentally Friendly: The remote control parts must be reused and recycled to minimize waste.
- Usability: The remote control must be easy to navigate by the user with minimal effort.
- Scalability: The remote control must be able to be modified and upgraded in the future to add more features.
- Robustness: The remote control must be able to respond accordingly to unexpected input from the user.
- Safety: The remote control engine must off automatically shut off the engine if it overheats

#### 2.2.3. Constraints

We then needed to take into consideration the constraints of our project.

Following are the constraints of the remote control starter:

- Engine must immediately turn off if its temperature is greater than 195° F [5]
- Trunk and car doors cannot remain open for more than 30 seconds.
- Engine cannot idle for longer than 120 seconds

- The design project should not have unnecessary latches.

## 3. Solution

We considered 3 possible solutions over the course of this design project, improving upon them iteratively until our final solution was reached with distinct advantages over the other two.

### 3.1. Solution 1

The first solution involved implementing the controller in two separate state diagrams, one to control the trunk and door locks, Fig 3.1.1, and the other to control the engine and heater, Fig 3.1.2. Development of this solution ceased once the development of the state diagram proved that we would need many states to have a fast input response speed while also having a long enough delay before the car automatically locks and automatically turns off the engine. In solutions 2 and 3 we improve on this solution by adding a common timer between the two states that triggers certain changes.

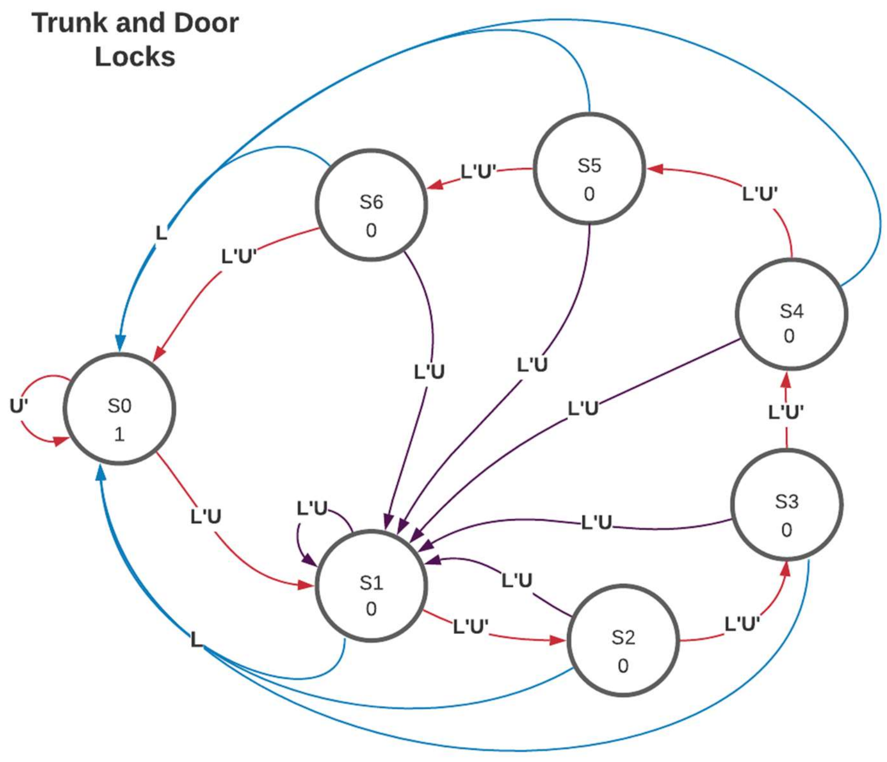
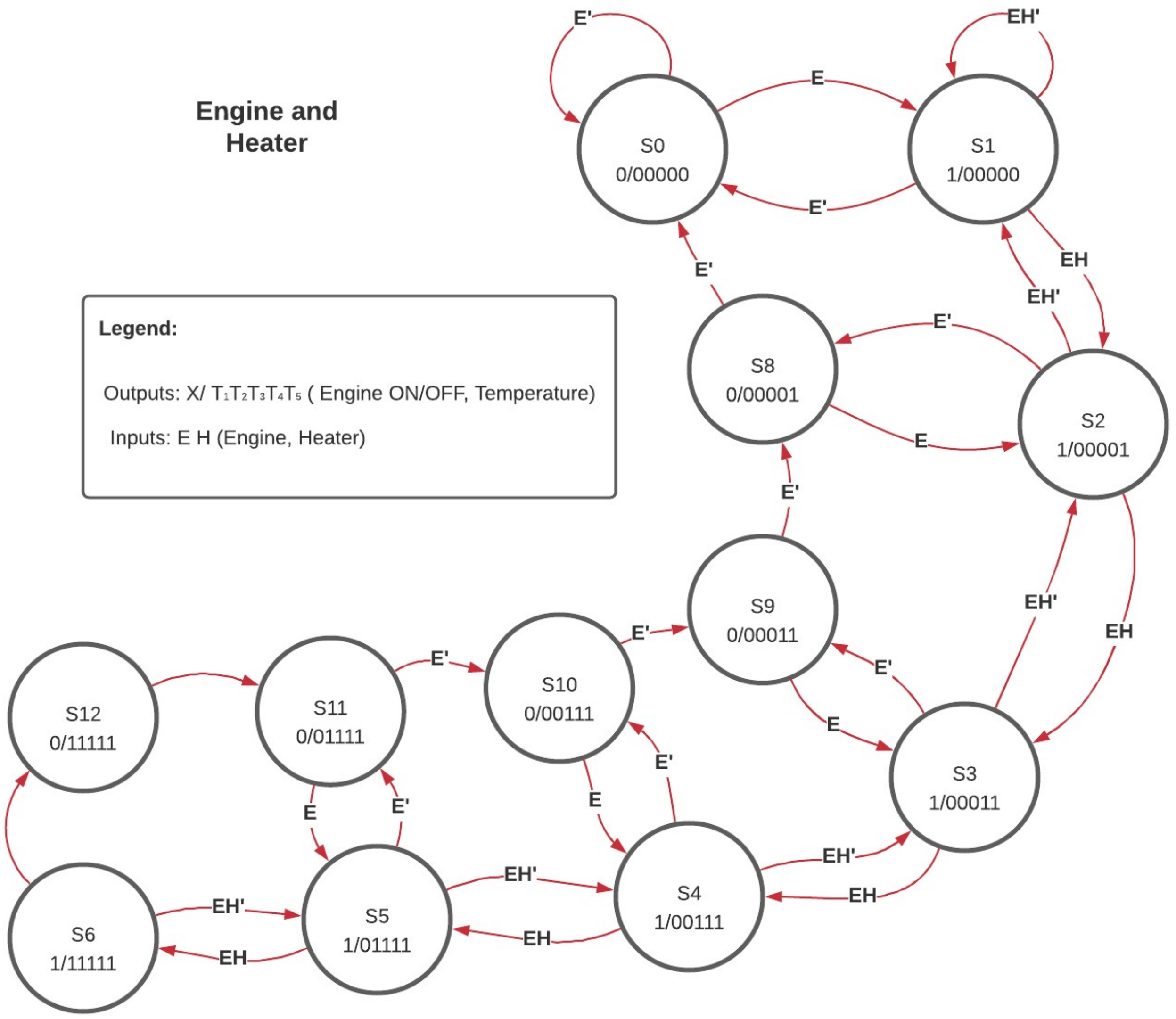

Trunk and Door Locks
S0
1
S1
0
S2
0
S3
0
S4
0
S5
0
S6
0
U'
L'U
L'U'
L
Legend:
Output: Z (1 = locked, 0 = unlocked)
Inputs: L U (Lock, Unlock)

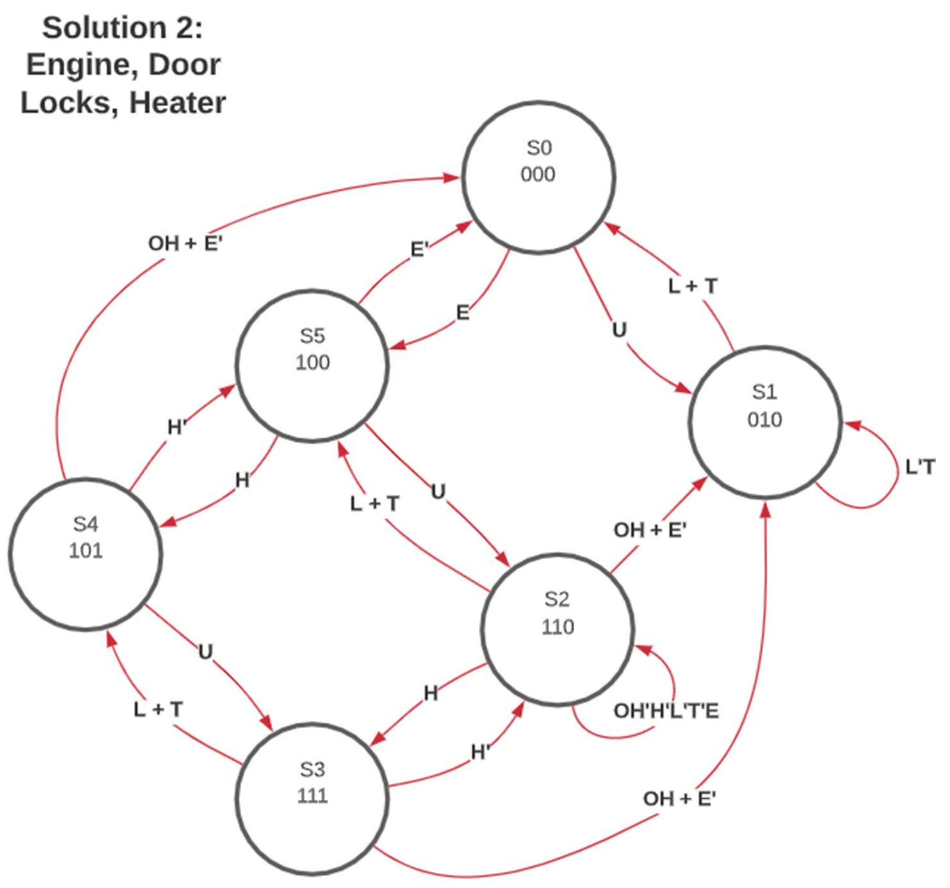
*Figure 3.1.1*: Solution 1 Trunk and Door Lock State Diagram



2. *Figure 3.1.2*: Solution 1 Engine and Heater State Diagram

## 3.2. Solution 2

Solution 2, as shown in Fig 3.2.1 combines our two state diagrams from solution 1. We also added a common counter to reduce the number of states while keeping input precision and the ideal time in each state. We moved to solution 3 from this solution because this solution does not have a way to shut the engine off after a certain period of time. It is also less modifiable due to having a highly interconnected timer.



**Legend:**

Outputs: Engine, Door Lock, Heater (Eng D H)

Inputs: Engine Start, Unlock, Lock, Overheat, Timer Done (E U L OH T)

*Figure 3.2.1*: Solution 2 State Diagram

## 3.3. Final Solution

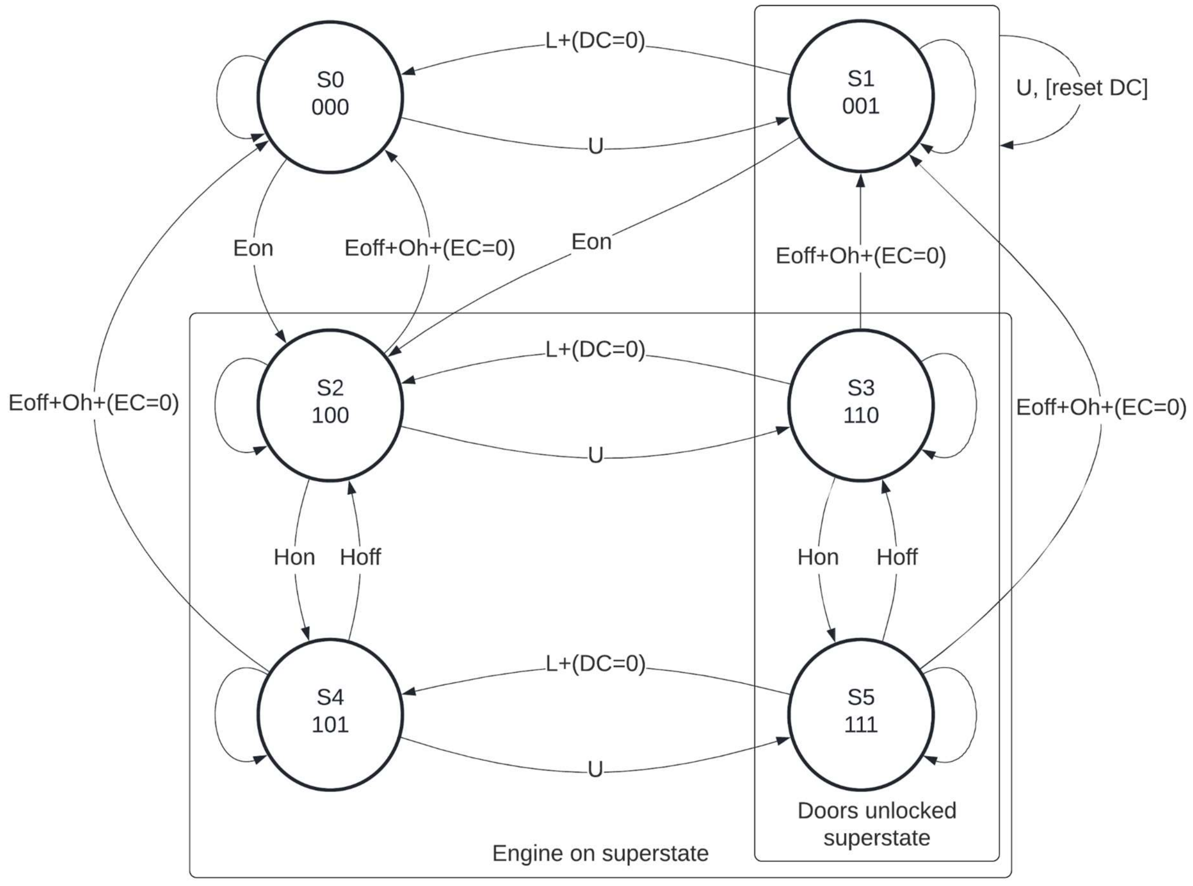
The final solution was developed specifically as an improvement to solution 2. It maintains the functionality of solution two while greatly simplifying the implementation. It includes user accessible inputs to turn the engine on and off, lock and unlock the doors, and turn the heater on and off from the remote. As well it displays the status of the doors engine and heater with indicator LEDs on the remote. The solution also implements a counter that shuts off the engine if it has been idling for more than two minutes. Major advantages in the simplicity of the implementation were achieved by using OR superstates to run the counters for the doors to lock, and the engine to shut off. These superstates enabled the two counters to be run simultaneously with only the implementation of a single state diagram.

This final solution is a fully Moore sequential machine. A Moore machine was chosen due to the simplicity of implementation and debugging over a Mealy machine for such a complex device [4]. VHDL implementation of the solution was shortened without the need for separate control of states and outputs The testing of the device was simplified since the output and the state were linked together so the outputs were easy to verify by testing what state the device was in, recording the output, and comparing the actual output to the expected; more information on testing methodology is in appendix 7.2, including a test bench used to verify outputs of each state in this manner.

This solution added the inputs engine off, heater on, and heater off as compared to solution 1; similar to solution 2 and for the same reason. These additional inputs enable the user to turn off the engine if they would like to. This functionality is very useful to the user if they start their car but then find they need to do something away from their car and it must be turned off. It also enables the engine to be turned off with the remote which is a crucial functionality missing in the previous solutions considered. As well this solution enables the user to control the heater independently of the engine. Due to how a car heater works [6] the heater can only run when the engine is running, since it pulls waste heat from the engine. Hence the heater can only be turned on from state 2 or 3, when the engine is running. It will proceed to either state 4 or 5 when the heater is turned on depending on the current status of the doors and trunk.

In this solution OR superstates were used to implement counters on the doors and trunk, as well as the engine. This enabled the counter to run whenever the machine is in any of the substates composing the superstate which provides a huge advantage in scalability over both previous solutions; as well, this simplified the VHDL implementation of this solution and made the code easier to maintain and provide future upgrades to. As well this change enables the counters to to avoid the need to be reset to their starting value on the entry of the superstate; or in the case of the doors unlocked superstate also when the unlock button is pressed while inside the superstate. Using the concept of overlapping superstates the system can be in either, both, or neither the engine on superstate and the doors on superstate [7]. This allows the counter for the doors to be running at the same time as the counter for the counter for the engine which was the reason why a similar implementation with two counters (one for the doors and trunk, and one for the engine) could not be made with the timers operating in the same manner as in solution 2. This also makes the counting of how long the engine has been on and how long the doors have been unlocked for more predictable timing than the previous solutions considered. This timing improvement made the solution more reliable than either previous solution considered. As well the improved counting predictability improved the scalability making it easier to add additional functionality to the superstates where counters were running, as well as additional superstates that implement counter functionality.

### 3.3.1. State diagrams and state table



Outputs: Engine, Doors unlocked, Heater

Inputs: Engine_on, Engine_off, Unlock, Lock, Heater_on, Heater_off, Overheat (Eon, Eoff, U, L,Hon, Hoff, Oh)

Counters: Engine_counter, Door_counter (EC, DC)

*Figure 3.3.1*: Solution 3 Simplified State Diagram

Figure 3.3.1 shows the simplified state diagram for the final solution, for clarity only the inputs that need to be high are shown on each transition; the formalized state diagram is figure ?? which shows the extra conditions necessary to ensure the machine enters the expected state. The machine is composed of two overlapping superstates, the engine on superstate and the doors unlocked superstate. Both superstates are OR superstates, and the machine can only be in one sub-state at once. When the engine on superstate is entered by turning the engine a counter is started that will count down from 120000 clock cycles (2 minutes) when this counter reaches 0 the engine superstate will be excited to either state 0 if the doors are locked, or state 1 if the doors are unlocked, this maintains if the system is either in or not in the doors unlocked superstate. When the machine enters the doors unlocked superstate a 30000 clock cycle (30 second) counter is started, unlike the engine counter this counter can be reset by pressing the unlock button on the controller again. When this happens the machine stays in the same substate and the doors unlocked counter begins to count down from 30000 clock cycles again. The unlabeled feedback on each state in this simplified diagram corresponds to the fact that if neither counter is 0 and no input that can change the current state is given the machine will stay in its present state. This functionality was implemented using an else statement in VHDL to ensure that all possibilities were covered.

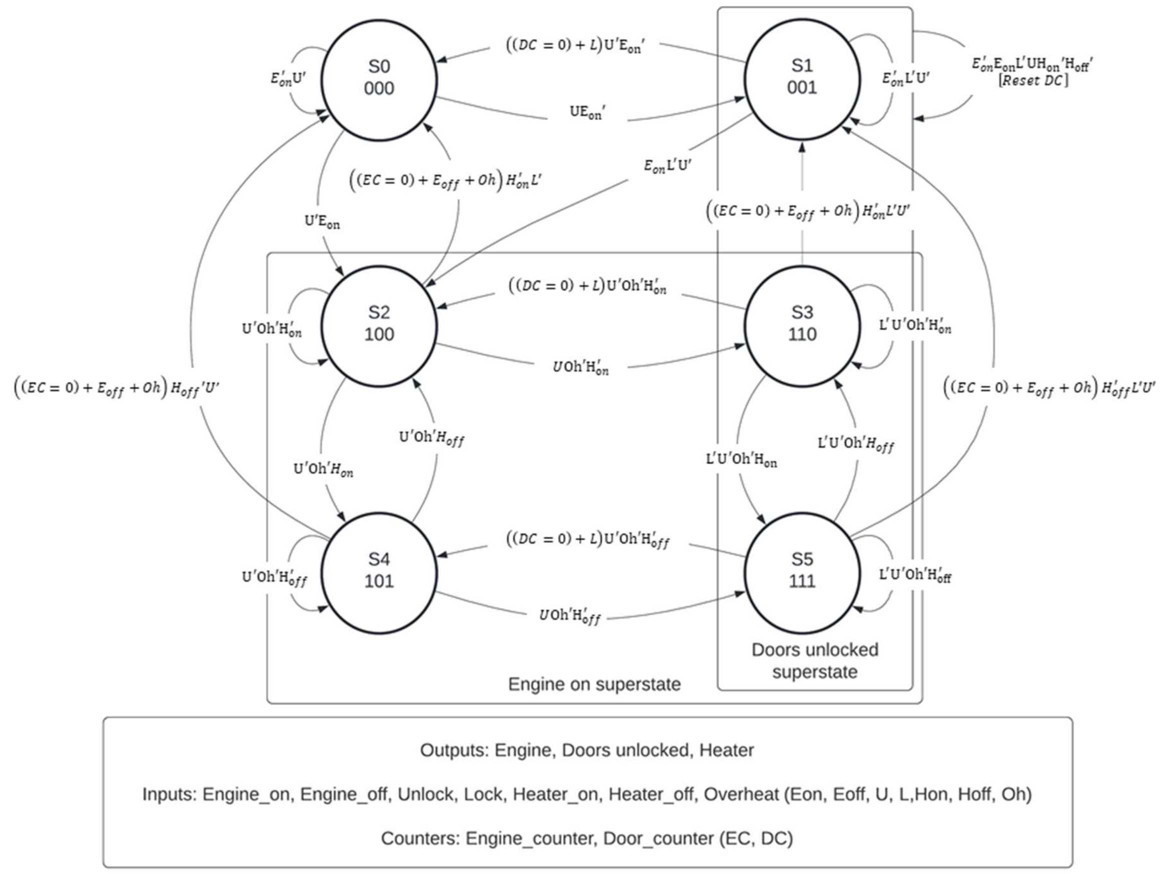

S0
000
S1
001
S2
100
S3
110
S4
101
S5
111
$E'_{on}U'$
$((DC = 0) + L)U'E_{on}'$
$UE_{on}'$
$E'_{on}L'U'$
$E'_{on}E_{on}L'UH_{on}'H_{off}'$ [Reset DC]
$((EC = 0) + E_{off} + Oh)H'_{on}L'$
$U'E_{on}$
$E_{on}L'U'$
$((EC = 0) + E_{off} + Oh)H'_{on}L'U'$
$((DC = 0) + L)U'Oh'H'_{on}$
$U'Oh'H'_{on}$
$UOh'H'_{on}$
$L'U'Oh'H'_{on}$
$((EC = 0) + E_{off} + Oh)H_{off}'U'$
$((EC = 0) + E_{off} + Oh)H'_{off}L'U'$
$U'Oh'H_{off}$
$U'Oh'H_{on}$
$L'U'Oh'H_{on}$
$L'U'Oh'H_{off}$
$((DC = 0) + L)U'Oh'H'_{off}$
$U'Oh'H'_{off}$
$UOh'H'_{off}$
$L'U'Oh'H'_{off}$
Doors unlocked superstate
Engine on superstate
Outputs: Engine, Doors unlocked, Heater
Inputs: Engine_on, Engine_off, Unlock, Lock, Heater_on, Heater_off, Overheat (Eon, Eoff, U, L,Hon, Hoff, Oh)
Counters: Engine_counter, Door_counter (EC, DC)

*Figure 3.3.2*: Solution 3 State Diagram

Figure 3.3.2 shows the proper state graph for the final solution, it has all the functionality of the simplified state graph previously shown, but it includes all conditions necessary for the state graph to be proper, ensuring that the system does not have an unknown state transition if multiple inputs are high at one time [8].

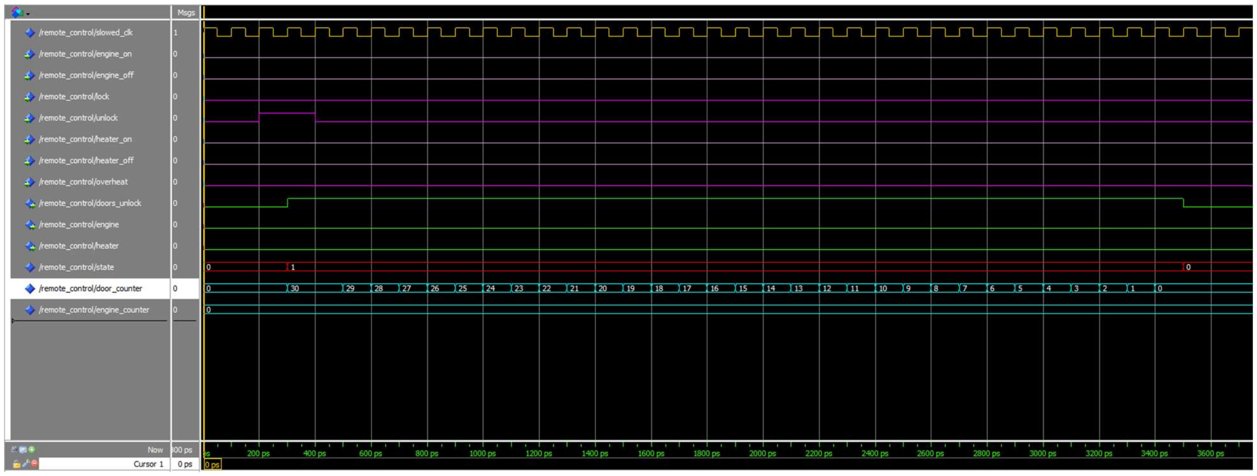
### 3.3.2. Solution selection

Table I: Decision matrix chart for the considered alternatives

| | | Solutions | | | | | |
|---|---|---|---|---|---|---|---|
| | | Solution 1 | | Solution 2 | | Final Solution | |
| **Criteria** | **Weight** | **Score** | **Partial Score** | **Score** | **Partial Score** | **Score** | **Partial Score** |
| **User control** | 0.1 | 5/10 | 0.05 | 8/10 | 0.08 | 8/10 | 0.08 |
| **Usability** | 0.15 | 7/10 | 0.105 | 6/10 | 0.09 | 6/10 | 0.09 |
| **Scalability** | 0.2 | 4/10 | 0.08 | 5/10 | 0.1 | 8/10 | 0.16 |
| **Reliability** | 0.1 | 3/10 | 0.03 | 7/10 | 0.07 | 8/10 | 0.08 |
| **Safety** | 0.15 | 4/10 | 0.06 | 7/10 | 0.105 | 7/10 | 0.105 |
| **Environment** | 0.1 | 4/10 | 0.04 | 6/10 | 0.06 | 8/10 | 0.08 |
| **Implementation** | 0.2 | 6/10 | 0.12 | 3/10 | 0.06 | 5/10 | 0.1 |
| **Sum** | 1.00 | | 0.485 | | 0.565 | | 0.695 |

The decision matrix used to select the final solution. The proposed solutions ranked on a number of criteria using from 0 to 10, then each criterion was assigned a weight based on the importance of it to the project. Using the overall scores of each solution our final solution was chosen because of its distinct advantages in implementation, scalability, and environmental factors specifically over solution 2 while maintaining large parts of the functionality. The implementation of superstates for counters enabled the device to run the two counters at once for the doors and engine and it made the implementation of the final solution simpler than solution 2, while making it more scalable for future functionality. This solution also had minor improvements in the reliability due to the further segmentation achieved using the superstates. While the final solution was not as simple to implement as solution 1 the it had several distinct advantages over it most notably reliability, by decreasing the number of states and simplification of the interconnections between them. When compared to solution 1 it was also notably better on environmental factors, due to the full implementation of user control of the engine shutoff, and a counter on the engine to prevent idling.

### 3.3.3. Waveforms



/remote_control/slowed_clk
/remote_control/engine_on
/remote_control/engine_off
/remote_control/lock
/remote_control/unlock
/remote_control/heater_on
/remote_control/heater_off
/remote_control/overheat
/remote_control/doors_unlock
/remote_control/engine
/remote_control/heater
/remote_control/state
/remote_control/door_counter
/remote_control/engine_counter
Now
Cursor 1

*Figure 3.3.3*: Waveform 1

Figure 3.3.3 shows an example of the system transitioning from state 0 to state 1 when the unlock button is pressed, on entry into state  which is a member substate of the doors unlocked superstate the door counter is started and the doors are unlocked. As time passes with no input given the counter ticks down until it reaches its final value, then the system leaves state one and enters into state 0 locking the doors. Right when the system enters into state 0 the unlock button is held, in doing so the counter is reset for one clock cycle do to this, this is because pressing the door unlock again in a state with the doors unlocked will reset the counter enabling the user to have more time with the doors unlocked if they would like.

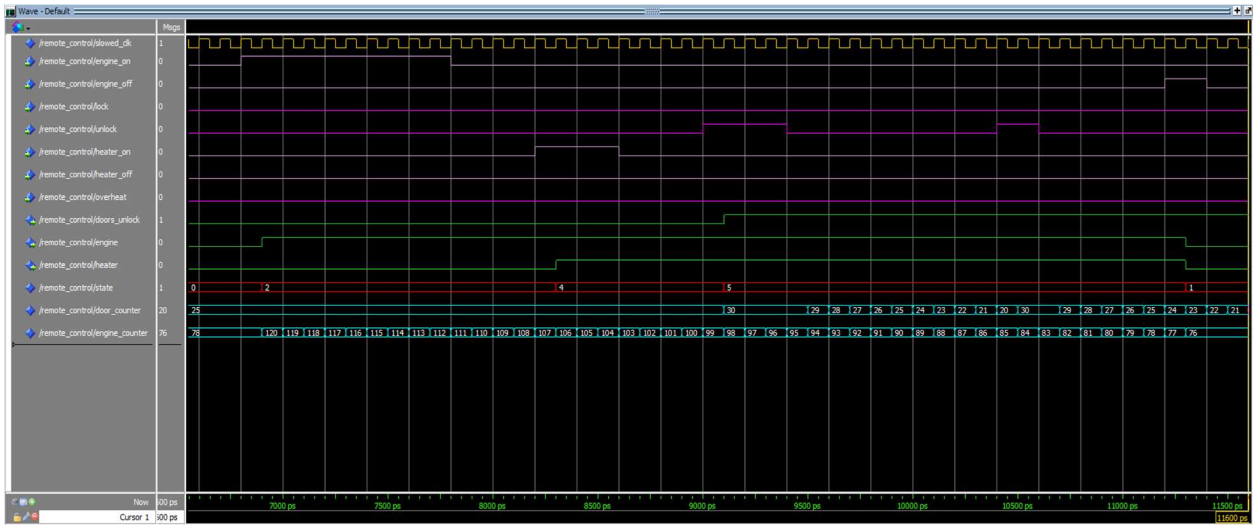


*Figure 3.3.4*: Waveform 2

Figure 3.3.4 shows an example of the system as it enters the engine on a superstate when the user pushes the engine on input the system moves from state 0 to state 2 and the engine counter begins. Looking at where the engine turns on even though the input stays high for several clock cycles the counter continues to count down, this is so the engine cannot be remotely operated for more than two minutes no matter the input. Once the machine is in the engine on superstate it can then be moved through some of the constituent states. Firstly, the system is moved to state 4 by turning the heater on, then the doors are unlocked moving the system into state 5 with the engine and heater on, and the doors unlocked. After some time has passed the door counter has gone down, when another unlock input is received the system stays in the current state, but the door counter is reset and begins counting down again. Finally, when the engine is turned off with the engine off input the system moves into state one which keeps the doors unlocked even though the engine and heater have both been shut down.

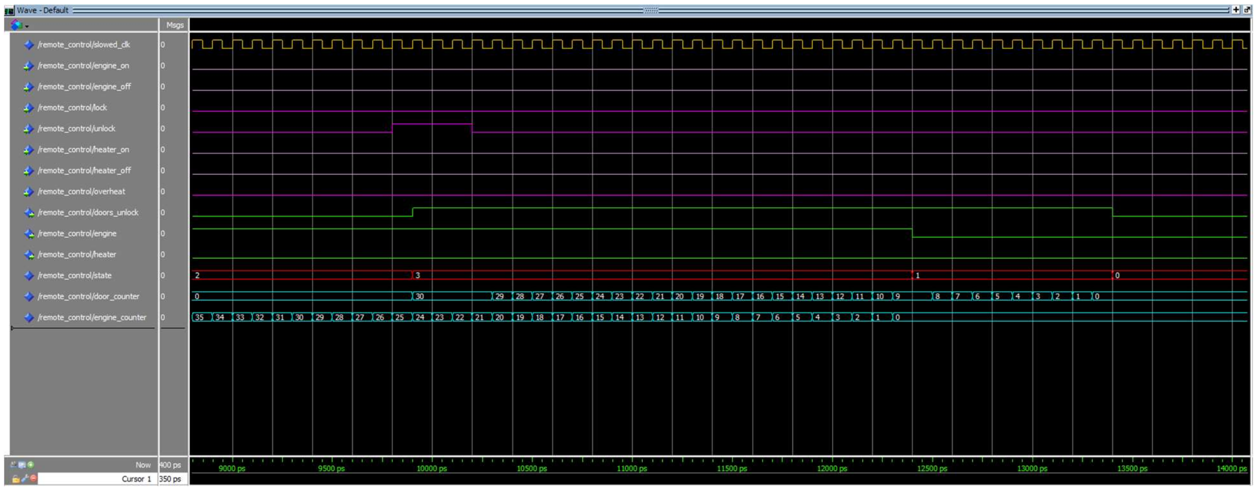


*Figure 3.3.5*: Waveform 3

Figure 3.3.5 shows the final example where we observe the functionality of the counters, the engine has been on for approximately 1.5 minutes at the start of this slide and is nearing shutting off the engine, before it does that the doors are unlocked and begin to also count down. When the engine timer completes the system keeps the doors unlocked and the counter continues running after the state transition until the system returns to state 0 with the doors locked and the engine off.

### 3.3.4. Features

The final solution implements all features of the previous solutions, as well as some others. On the remote there are lock and unlock switches that control the trunk and doors; switches to start and stop the engine; and switches to turn the heater on and off. When an overheat alarm is triggered the engine will shut down. Indicator LEDs on the remote show the status of the doors and trunk, engine, and heater. Engine will shut down if it has been running for longer than 120 seconds. The doors and trunk will lock if they have been unlocked for more than 30 seconds since the last unlock input. Starting the engine automatically locks the doors and trunk.

Table II: Mapping of inputs to FPGA switches

| Input | Input Switch Used |
|---|---|
| unlock | SW0 |
| lock | SW1 |
| engine_on | SW2 |
| engine_off | SW3 |
| heater_on | SW4 |
| heater_off | SW5 |
| overheat | SW6 |

As this project is to design a prototype for a remote car starter and implement it on the FPGA we used the switches of our FPGA for this course (Basys 3 Artix-7 FPGA Board) as inputs for testing. The table above lists which of these inputs was mapped to which switch on that board. The inputs unlock, lock, engine_on, engine_off, heater_on, and heater off all correspond to inputs on the user remote. The input overheat corresponds to an internal overheat alarm that is triggered if the engine exceeds its normal operating temperature.

Table III: mapping of outputs to FPGA LEDs

| Output | Output LED Used |
|---|---|
| Doors and trunk unlocked | LED0 |
| Engine on | LED2 |
| Heater on | LED4 |

The outputs of the system go to indicator LEDs as mapped in the above table; these outputs would also be the output of the remote that would signal the car to do the appropriate actions. These specific LEDs were used since they fall above the corresponding inputs. LED0 is above SW) which unlocks the doors and trunk, LED2 is above SW2 which turns on the engine, and LED4 is above SW4 which turns the heater on. This enhances their functionality as indicator lights, and it is something we hope to continue through the development of the device.

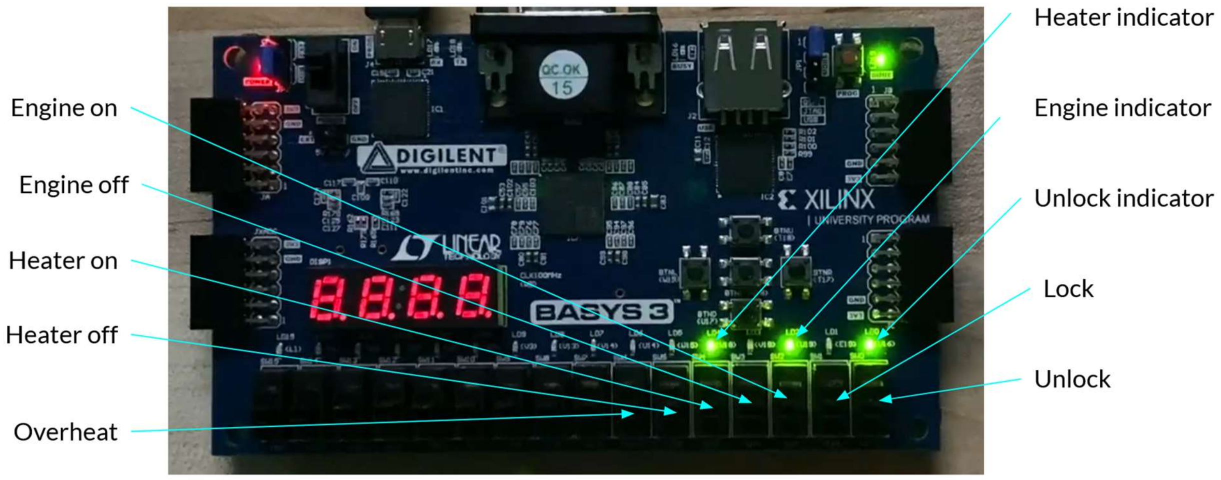


*Figure 3.3.6*: Inputs and Outputs

Figure 3.3.6 shows the inputs and outputs of the final solution on the Basys 3 Artix-7 FPGA Board. Switches SW0 to SW6 were used for inputs, See table III for the specific mapping of the switches to the inputs. LEDs LED0, LED2, and LED4 were used for outputs, see table IV for mapping of features to sequence of inputs.

Table IV: Inputs and Outputs

| **Feature** | **Component** |
|---|---|
| Unlocking and locking doors and trunk | Switch 0, Switch 1, and LED 0 |
| Starting and shutting off engine | Switch 2, Switch 3, and LED 2 |
| Heater starting and shutting off | Switch 4, Switch 5, and LED 4 |
| Engine shut off on overheat | Switch 6 |

This table shows the features and the components on the FPGA used to fulfill them.

### 3.3.5. Environmental, Societal, Safety, and Economic Considerations

The following table summarizes the environmental, societal, safety, and economic considerations we gave attention to in making the remote car starter.

Table V: Environmental, Societal, Safety, and Economic Considerations

| Considerations | Features |
|---|---|
| Environmental | - The components used for the controller are long lasting and can be reused and recycled, hence reducing the amount of waste produced.<br>- The components cost little power, hence reducing power usage.<br>- The controller turns off the engine automatically after 2 minutes idle, this reduces energy wastage and minimizes emissions of pollutants. |
| Societal | - The controller has easy to navigate switches, making it user friendly.<br>- The remote controller works 24/7, 365 days without issues, ensures that the car is working properly and under control all the time. This essentially makes sure accidents are minimized.<br>- The size of the controller is small, which helps the user to carry it with them in their pocket. |
| Safety | - The controller can turn off the engine automatically when it heats up. This minimizes the chance of the engine blowing up.<br>- The trunk and door closes automatically after 30 seconds to prevent unauthorized access to the car and theft.<br>- The components are sealed properly to make sure it is waterproof, and prevents electrical shocks. |

| | |
|---|---|
| | - Controller operates on low voltage to further reduce shock risk |
| Economical | - The components used are inexpensive but fulfills the user requirements and is reliable.<br>- The automatic engine turn off feature saves gas and is economically beneficial in the long run.<br>- The controller parts can be recycled and modified for future update, hence saves money in the long run. |

### 3.3.6. Limitations

The remote car starter was just a prototype and was limited in various ways. We could only use the components available on the FPGA board to work towards our goal. Our design project was limited within the FPGA components and VHDL coding. We wanted to use a heat sensor for automatic engine shut off when a certain degree of heat is detected. We also had the idea of including a buzzer to beep when actions such as the door lock, engine on/off is completed. However, none of the mentioned features could be implemented due to lack of sensors on the FPGA board or an actual car to connect and test on. We could only use buttons, switches, and LEDs to reach our project goal. We used switches because it is easier to implement than buttons. The solutions had to be modeled accordingly to fit the mentioned limitations.

## 4. Team Work

Our team had regular meetings to discuss the progress of the project. During each meeting we tracked the amount of work done and discussed individual tasks with each other to make sure every member is on the same page. We split the workload equally and set a deadline for each task. Finally, we set a date for the next meeting for further discussions. Our interaction was not only limited to face-to-face team meetings but also through online communication channels.

### 4.1. Meeting 1

Time: November 1, 2021, 5:15 pm to 6:15 pm
Meeting minutes: 60 minutes
Agenda: Distribution of project tasks, and discussion of problem

| Team Member | Previous Task | Completion State | Next Task |
|---|---|---|---|
| **Nazmul Hoque** | N/A | N/A | Design Problem and requirement specifications |

| | | | |
|---|---|---|---|
| **Samuel Elgert** | N/A | N/A | State Graphs for doors and trunks |
| **Kieren O'Neil** | N/A | N/A | Introduction and solution brainstorming |

3.

## 4.2. Meeting 2

Time: November 18, 2021, 10:00 am to 10:30 am
Meeting minutes: 30 minutes
Agenda: Distribution of project tasks, and discussion of problem and progress

| Team Member | Previous Task | Completion State | Next Task |
|---|---|---|---|
| **Nazmul Hoque** | Design Problem and requirement specifications | 80 % | Features |
| **Samuel Elgert** | State Graphs for doors and trunks | 80 % | State Graphs for engine and heater |
| **Kieren O'Neil** | Introduction and solution brainstorming | 80 % | VHDL implementation of state diagrams |

## 4.3. Meeting 3

Time: November 25, 2021, 10:00 am to 10:30 am
Meeting minutes: 30 minutes
Agenda: Distribution of project tasks, and discussion of problem and progress

| Team Member | Previous Task | Completion State | Next Task |
|---|---|---|---|
| **Nazmul Hoque** | Features | 80% | Environmental, Social, Safety, Economical considerations |
| **Samuel Elgert** | State Graphs for engine and heater | 80% | Compare and finalize iteratives |
| **Kieren O'Neil** | VHDL implementation of state diagrams | 80 % | Finalize solution |

### 4.4. Meeting 4

Time: November 29, 2021, 5:30 pm to 6:00 pm
Meeting minutes: 30 minutes
Agenda: Discussion of project progress, and prepare for presentation

| Team Member | Previous Task | Completion State | Next Task |
|---|---|---|---|
| **Nazmul Hoque** | Environmental, Social, Safety, Economical considerations | 100% | Limitations, Conclusion and Future works |
| **Samuel Elgert** | Compare and finalize iteratives | 90% | Poster |
| **Kieren O'Neil** | Finalize solution | 90 % | Final report adjustments |

## 5. Conclusion and Future Work

Throughout the design project, we utilized different components available on the Xilinx Basys 3 FPGA Board. We tested the remote-control car starter by implementing VHDL code developed using state graphs and behavioral modelling. The use of behavioral modelling was inevitable as it was easier to implement than any other data modelling on this complex system. Our project reached its goal, as it can automatically open and close the trunk and doors. It is also able to start and turn off the engine automatically. The remote control operated properly taking environmental, societal, safety, and economical aspects into consideration. The different features were accomplished with some unavoidable limitations. We have taken scalability, modifiability, and other engineering attributes into account for future design improvements. Involving different sensors and utilizing them to modify the remote car starter is our biggest future objective. Implementing a car alarm to prevent unauthorized access, using a heat sensor for automatic engine turn off, and task completion beeps are some major options. We will also look at options of including buttons rather than switches for inputs to reduce complexity. Using the 7-segment display for displaying different values such as time, weather, temperature, and humidity will also be taken into consideration. We will not limit ourselves within the FPGA board but will take the remote control to the next level by connecting it to our smartphone and have features for tracking and finding the controller if it gets lost. There is large room for improvement in the final solution and the benefits of having a remote car starter are endless.

## 6. References

[1] J. Dargay and D. Gately, “Vehicle ownership to 2015: Implications for energy use and emissions,” *Energy Policy*, vol. 25, pp. 1121-1127, December 1997. [Online]. Available: https://reader.elsevier.com/reader/sd/pii/S0301421597001043?token=FBFF7DC519A639D89669C3A0A81D486C6775D511E1B1580F953A93042B744DD29C89C6CDD4486A54334B3EB2EF434755&originRegion=us-east-1&originCreation=20211104185708

[2] Environment and Climate Change Canada, “Temperature and Precipitation Graph for 1981 to 2010 Canadian Climate Normals EDMONTON INT'L A ,” *Canadian Climate Normals 1981-2010 Station Data*, 16-Nov-2021. [Online]. Available: https://climate.weather.gc.ca/climate_normals/results_1981_2010_e.html?searchType=stnName&txtStationName=edmonton&searchMethod=contains&txtCentralLatMin=0&txtCentralLatSec=0&txtCentralLongMin=0&txtCentralLongSec=0&stnID=1865&dispBack=0. [Accessed: 18-Nov-2021].

[3] C.S. Cheng, H. Auld, Q. Li, and G. Li, “Possible impacts of climate change on extreme weather events at local scale in south–central Canada,” *Climatic Change,* vol. 112, pp. 963-979, September 2011. [Online]. Available: https://link.springer.com/article/10.1007/s10584-011-0252-0

[4] C. H. Roth and L. K. John, *Digital Systems design using VHDL*. Mason, OH: Cengage Learning, 2016.

*[5] What Happens When Your Engine Overheats? | Rick Hendrick ... https://www.hendrickatlanta.com/service/service-tips/happens-engine-overheats/.*

[6] Pirotais, F., et al. “A Model of Energetic Interactions between a Car Engine, the Cabin Heating System and the Electrical System.” *A Model of Energetic Interactions Between a Car Engine, the Cabin Heating System and the Electrical System*, 9 July 2002, https://www.sae.org/publications/technical-papers/content/2002-01-2224/.

[7] “Embedded System Design.” *Google Books*, Google, https://books.google.ca/books?id=0BQ7WkOa_28C&pg=PA82&lpg=PA82&dq=or%2Bsuperstate&source=bl&ots=Zh6bw_6G4w&sig=ACfU3U07mnVszp0pvmt4JgZNvJ_dpb1uFQ&hl=en&sa=X&ved=2ahUKEwjh8LTk-sD0AhUbnWoFHdmsBCEQ6AF6BAghEAM#v=onepage&q&f=false.

[8] “Digital Design 4th Edition - Morris Mano.pdf.” *Google Drive*, Google, https://docs.google.com/file/d/0B8-drkZsESDnN2NmYTQxYjQtYTMwZi00N2IzLTkxNjgtZjI1NTZiN2FjNDli/edit.

[9] Waqar, Omer. “TRU Moodle: CENG 3010.” *TRU Moodle: CENG 3010*, https://moodle.tru.ca/course/view.php?id=34670.

[10] “Basys 3 Artix-7 FPGA Board.” *Xilinx*, https://www.xilinx.com/products/boards-and-kits/1-54wqge.html.

[11] “BASYS™3 Artix-7 FPGA Board.” *DigiKey*, https://www.digikey.ca/en/product-highlight/d/digilent/basys3-artix-7-fpga-board.

# 7. Appendix

## 7.1. VHDL Code

The VHDL implementation of this project was initially completed in the Intel® FPGA edition of ModelSim using Notepad++ as an editor initially. The implementation was tested using simulation in ModelSim (see appendix 7.2 for more information). Following testing we moved to Xilinx Vivado for implementation on the Xilinx Basys 3 FPGA, final issues were fixed in Vivado throughout this process.

The implementation was composed of two entities, a clock divider, and the controller. The clock divider was modified from the implementation in lab 8, the main change was to switch to a 1 kHz clock frequency. The remote control was implemented in one VHDL entity which could be done due to the advantages of the superstates used in the final solution. Note that this code had to be reformatted to fit into the format of this report.

This implementation eliminated unnecessary latches on everything. There is a necessary inferred latch by Vivado present on the slowed clock due to the operation of the clock divider, this is due to the implementation of the clock divider and this cannot be eliminated without changing the clock divider. As well there are inferred latches on both counters in order to have them properly update on each clock cycle. Other latches that are not inferred by Vivado are present to the use of if-else statements which are necessary for the machine to operate with the superstates properly. In the future The number of these required latches will be reduced in future iterations of this design.

The constraints file that was used for mapping the inputs and outputs of this code was modified from the one listed on the moodle page for this course (CENG 3010) [9].

### 7.1.1. Clock Divider

```
-- clock divider
-- generates 1 kHz clock signal from the 100 MHz clock of FPGA
-- modified from clock divider in lab 8
-- for CENG 3010 project

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- create entity
entity clock_divider is
        port(in_clk: in std_logic;
         slow_clk: inout std_logic);
end clock_divider;

-- create architecture
architecture Behavioral of clock_divider is

-- time for 100000 clock cycles between inversion
```

```
constant N: integer := 100000;
signal counter: integer range 1 to N;

begin

        process (in_clk, counter, slow_clk)
  begin
        -- increment counter
         if (in_clk'event and in_clk = '1') then
                counter <= counter + 1;
         end if;

        -- if counter is reached invert clock
         if counter = N then
                slow_clk <= not slow_clk;
                counter <= 1;
         end if;

  end process;

end Behavioral;
```

### 7.1.2. Controller

```
-- remote car stater
-- implements remote control of a car
-- for CENG 3010 project

library IEEE;

-- using standard logic type for variables
use IEEE.STD_LOGIC_1164.ALL;
-- using integers for states, and counter
use IEEE.NUMERIC_STD.ALL;

entity remote_control is
        Port(engine_on, engine_off, lock, unlock, heater_on, heater_off, overheat, clk: in std_logic;
        -- remove comment for state output to test
        --test: out std_logic_vector (5 downto 0);
         doors_unlock, engine, heater: out std_logic);
end remote_control;

architecture behavioral of remote_control is

signal slowed_clk: std_logic;
signal state, nextstate: integer range 0 to 5;

-- leave doors unlocked for 30000 slowed clock cycles (30 seconds)
constant door_count_for: integer := 30000;
signal door_counter, next_door_counter: integer range 0 to door_count_for;

-- leave engine running for 120000 slowed clock cycles (120 seconds)
```

```
constant engine_count_for: integer := 120000;
signal engine_counter, next_engine_counter: integer range 0 to engine_count_for;

-- import a clock divider
component clock_divider
   port(in_clk: in std_logic;
             slow_clk: inout std_logic);
end component clock_divider;

begin
   -- instantiate a clock divider
   -- convert clock from 100 MHz to 1 kHz
   divider_1 : clock_divider port map(in_clk => clk,
                        slow_clk => slowed_clk);

            -- Process triggers on any input, state, and the 1 kHz clock
   process(state, engine_on, engine_off, lock, unlock, heater_on, heater_off, overheat, slowed_clk, door_counter,
engine_counter)
   begin
             case state is

                        -- complete and proper
                        -- state 0
                         -- engine off doors locked
                         when 0 =>
                                    engine <= '0';
                                    doors_unlock <= '0';
                                    heater <= '0';
                                    if (engine_on = '1' and unlock = '0') then
                                             nextstate <= 2;
                                             next_engine_counter <= engine_count_for;
                                    elsif (engine_on = '0' and unlock = '1') then
                                             next_door_counter <= door_count_for;      -- reset door counter
                                             nextstate <= 1;
                                    else nextstate <= 0;
                                    end if;

                         -- complete and proper
            -- superstate 1
                         -- engine off doors unlocked heater off door_counter
                        when 1 =>
                                    engine <= '0';
                                    doors_unlock <= '1';
                                    heater <= '0';
                                    if (engine_on = '1' and lock = '0' and unlock = '0') then
                                             nextstate <= 2;
                                             next_engine_counter <= engine_count_for;
                                    elsif (door_counter = 0 or (engine_on = '0' and lock = '1' and unlock = '0')) then
nextstate <= 0;
                                    elsif (engine_on = '0' and lock = '0' and unlock = '1') then
                                             nextstate <= 1;
                                             next_door_counter <= door_count_for;      -- reset door counter
                                    else
```

```
                    next_door_counter <= door_counter - 1;      -- iterate on door counter
                    nextstate <= 1;
                end if;

        -- complete and proper
        -- superstate 2
        -- engine on doors locked heater off engine_counter
        when 2 =>
                engine <= '1';
                doors_unlock <= '0';
                heater <= '0';
                next_engine_counter <= engine_counter - 1;
                if (engine_counter = 0) or (overheat = '1' and heater_on = '0' and unlock = '0') or
(engine_off = '1' and heater_on = '0' and unlock = '0') then
                    nextstate <= 0;
                elsif (overheat = '0' and heater_on = '1' and unlock = '0' and engine_off = '0') then
nextstate <= 4;
                elsif (overheat = '0' and heater_on = '0' and unlock = '1' and engine_off = '0') then
                    next_door_counter <= door_count_for;        -- reset door counter
                    nextstate <= 3;
                else nextstate <= 2;
                end if;

        -- complete and proper
        -- superstate 3
        -- engine on doors unlocked heater off door_counter, engine_counter
        when 3 =>
                engine <= '1';
                doors_unlock <= '1';
                heater <= '0';
                next_engine_counter <= engine_counter - 1;
                if (engine_counter = 0) or (overheat = '1' and engine_off = '0' and lock = '0' and unlock =
'0' and heater_on = '0')
                or (overheat = '0' and engine_off = '1' and lock = '0' and unlock = '0' and heater_on = '0')
                    then nextstate <= 1; --  keep current door counter value by not resetting it
                elsif (door_counter = 0 or (overheat = '0' and engine_off = '0' and lock = '1' and unlock =
'0' and heater_on = '0')) then nextstate <= 2;
                elsif (overheat = '0' and engine_off = '0' and lock = '0' and unlock = '0' and heater_on =
'1') then nextstate <= 5;
                elsif (overheat = '0' and engine_off = '0' and lock = '0' and unlock = '1' and heater_on =
'0') then
                    nextstate <= 3;
                    next_door_counter <= door_count_for;
                else
                    next_door_counter <= door_counter - 1;      -- interate on door counter
                    nextstate <= 3;
                end if;

        --
        -- superstate 4
        -- engine on doors locked heater on engine_counter
        when 4 =>
                engine <= '1';
```

```
                                        doors_unlock <= '0';
                                        heater <= '1';
                                        next_engine_counter <= engine_counter - 1;
                                        if (engine_counter = 0) or (overheat = '1' and engine_off = '0' and heater_off = '0' and
unlock = '0')
                                        or (overheat = '0' and engine_off = '1' and heater_off = '0' and unlock = '0')
                                                    then nextstate <= 0;
                                        elsif (overheat = '0' and engine_off = '0' and heater_off = '1' and unlock = '0') then
nextstate <= 2;
                                        elsif (overheat = '0' and engine_off = '0' and heater_off = '0' and unlock = '1') then
                                                    next_door_counter <= door_count_for;        -- reset door counter
                                                    nextstate <= 5;
                                        else nextstate <= 4;
                                        end if;

                        -- superstate 5
                        -- engine on doors unlocked heater on door_counter, engine_counter
                        when 5 =>
                                        engine <= '1';
                                        doors_unlock <= '1';
                                        heater <= '1';
                                        next_engine_counter <= engine_counter - 1;
                                        if (engine_counter = 0) or (overheat = '1' and engine_off = '0' and  overheat = '0' and
lock = '0' and unlock = '0' and heater_off = '0')
                                        or (overheat = '0' and engine_off = '1' and  overheat = '0' and lock = '0' and unlock = '0'
and heater_off = '0')
                                                    then nextstate <= 1; -- keep current door counter value by not resetting it
                                        elsif (door_counter = 0) or (overheat = '0' and engine_off = '0' and  overheat = '0' and
lock = '1' and unlock = '0' and heater_off = '0') then nextstate <= 4;
                                        elsif (overheat = '0' and engine_off = '0' and  lock = '0' and unlock = '0' and heater_off =
'1') then nextstate <= 3;
                                        elsif (overheat = '0' and engine_off = '0' and lock = '0' and unlock = '1' and heater_off =
'0') then
                                                    nextstate <= 5;     -- reset door counter
                                                    next_door_counter <= door_count_for;
                                        else
                                                    next_door_counter <= door_counter - 1;      -- iterate on door counter
                                                    nextstate <= 5;
                                        end if;

            end case;
  end process;

  -- update state, door counter, and engine counter on rising edge of clock
  process (slowed_clk)
  begin
            if slowed_clk = '1' and slowed_clk'event then
                        state <= nextstate;
                        door_counter <= next_door_counter;
                        engine_counter <= next_engine_counter;
            end if;
  end process;
```

```
   -- displays current state on leds for debugging
   -- remove comment to test
   -- ensure corresponding comments above and in constraints are also removed
--   process (state)
--   begin
--          if state = 0 then test <= "000001";
--          elsif state = 1 then test <= "000010";
--          elsif state = 2 then test <= "000100";
--          elsif state = 3 then test <= "001000";
--          elsif state = 4 then test <= "010000";
--          elsif state = 5 then test <= "100000";
--          end if;
--   end process;

end behavioral;
```

## 7.1.3. Constraints file used

```
## Constraints file for CENG 3010 final project
## modified from Basys3_master.xdc on CENG 3010 moodle page
## unnecessary commented outputs removed

## sets config voltage
## eliminates config voltage error
set_property CFGBVS VCCO [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]

## Clock signal
## 100 MHz default clock
set_property PACKAGE_PIN W5 [get_ports clk]
   set_property IOSTANDARD LVCMOS33 [get_ports clk]
   create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

## Switches
set_property PACKAGE_PIN V17 [get_ports {unlock}]
   set_property IOSTANDARD LVCMOS33 [get_ports {unlock}]
set_property PACKAGE_PIN V16 [get_ports {lock}]
   set_property IOSTANDARD LVCMOS33 [get_ports {lock}]
set_property PACKAGE_PIN W16 [get_ports {engine_on}]
   set_property IOSTANDARD LVCMOS33 [get_ports {engine_on}]
set_property PACKAGE_PIN W17 [get_ports {engine_off}]
   set_property IOSTANDARD LVCMOS33 [get_ports {engine_off}]
set_property PACKAGE_PIN W15 [get_ports {heater_on}]
   set_property IOSTANDARD LVCMOS33 [get_ports {heater_on}]
set_property PACKAGE_PIN V15 [get_ports {heater_off}]
   set_property IOSTANDARD LVCMOS33 [get_ports {heater_off}]

## overheat is set as a switch for testing, will become an overheat alarm on engine
set_property PACKAGE_PIN W14 [get_ports {overheat}]
   set_property IOSTANDARD LVCMOS33 [get_ports {overheat}]

## LEDs
## LEDs above the on doors_unlock, engine_on, and heater_on
set_property PACKAGE_PIN U16 [get_ports {doors_unlock}]
   set_property IOSTANDARD LVCMOS33 [get_ports {doors_unlock}]
set_property PACKAGE_PIN U19 [get_ports {engine}]
   set_property IOSTANDARD LVCMOS33 [get_ports {engine}]
set_property PACKAGE_PIN W18 [get_ports {heater}]
   set_property IOSTANDARD LVCMOS33 [get_ports {heater}]

## state LED outputs
## remove comment for testing
#set_property PACKAGE_PIN W3 [get_ports {test[0]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {test[0]}]
#set_property PACKAGE_PIN U3 [get_ports {test[1]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {test[1]}]
#set_property PACKAGE_PIN P3 [get_ports {test[2]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {test[2]}]
#set_property PACKAGE_PIN N3 [get_ports {test[3]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {test[3]}]
#set_property PACKAGE_PIN P1 [get_ports {test[4]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {test[4]}]
#set_property PACKAGE_PIN L1 [get_ports {test[5]}]
#    set_property IOSTANDARD LVCMOS33 [get_ports {test[5]}]
```

## 7.2. Testing Methodology

### 7.2.1. Process

The implementation was tested throughout the process of completing it, since different software with different testing abilities was used throughout work on solution implementation, the testing process that was used to confirm operation of the solution implementation is listed here for completeness.

Firstly the initial implementation of all tested solutions was completed in the Intel® FPGA edition of ModelSim. When the implementation was complete It was tested thoroughly in using the simulation capabilities of that ModelSim. Due to the simulation speed of ModelSim the amount the clock was slowed by the clock divider was changed to one order of magnitude, and the timer was changed from 30000 clock cycles to 30. First the transitions between states were verified using the waveform view and by forcing signals. As well for the solutions that required a counter (2 and 3) the operation of the counter was tested by entering the counter states (in solution 2) and the counter OR superstates (in solution 3); when in a state with an active counter it was verified that the counter was operating as intended and counting down the number of clock cycles it would stay in that state for if no breakout input was received. Some selected waveforms output from this testing are shown in the final solution section to demonstrate the functionality of the final solution. These waveforms were generated in this testing phase of the implementation of the final solution.

If the waveform simulation confirmed that states were transitioning properly the next stage was to check outputs at each state. A test bench using a constant array listing the values of each output at a state in the address in the array of that state. The test bench would output an error if any output did not match its expected value for that state. Since the counter states and OR superstate counters all have the same output and the same value of the state variable they are treated as one state for the purposes of this test. The final test bench that was used in this project to test the final solution is shown in 7.2.2 along with its output.

Once the preceding steps were complete the software used changed to Xilinx Vivado where the solution would be physically implemented on the FPGA. Firstly, the solution input into a VHDL project and the counter and clock were changed to their real-world values from the reduced values used for ModelSim simulation. An additional output used for testing was put into the code which outputs the current state of the device on the LEDs; for example, the final solution has six states which were output on LEDs 10 to 15 with LED - 10 being the current state displayed in a one-hot method. Then the standard steps of Vivado workflow were followed; first synthesis was completed, then errors, critical warnings and warnings from synthesis were fixed. Then implementation was completed, errors, critical warnings and warnings from that process were fixed. Finally the bitstream was generated and again errors, critical warnings and warnings from it were fixed. Once a Bitstream had been generated the FPGA was programmed using it.

With the device now implemented on the FPGA the previous testing needed to be repeated to confirm that state transitions and outputs were working properly. A selection of state transitions were checked to confirm operation as intended, due to this functionality already being tested in ModelSim an exhaustive test was deemed unnecessary and only a subset of possible state transitions. Once states were checked the output for each state was verified by recording the state and output and comparing it to what was expected; once output was verified testing was completed, the output of the state was disabled and the device was fully functional.

### 7.2.2. Test bench used to test final solution

```
-- test bench
library IEEE;
use IEEE.std_logic_1164.all;

entity tester is
   Port(clk: in std_logic);    -- clk created seperatley
end tester;

architecture test1 of tester is
-- instantiate a remote to test
component remote_control is
        Port(engine_on, engine_off, lock, unlock, heater_on, heater_off, overheat, clk: in std_logic;
        -- remove comment for state output to test
        --test: out std_logic_vector (5 downto 0);
        doors_unlock, engine, heater: out std_logic);
end component;


-- testing output of each state, 6 states (0-5)
constant N: integer := 5;

-- create array types to hold inputs and outputs
type std_arr1 is array(0 to N) of std_logic_vector(0 to 2);
type std_arr2 is array(0 to N) of std_logic_vector(0 to 6);

-- create lookup arrays and nesecary signals
constant output_arr1: std_arr1 :=
("000","001","100","110","111","101");

constant input_arr2: std_arr2 :=
("0001000","1000000","0001000","0000010","0010000","0000000");

signal inputs: std_logic_vector(0 to 6);
signal outputs: std_logic_vector(0 to 2);
--signal test_state: std_logic_vector(5 downto 0);

begin
  process
  begin
        for i in 1 to N loop -- loop for N tests
        --outputs <= output_arr1(i);
        inputs <= input_arr2(i);
        wait for 20 ns;
        -- always report test interaction
        report integer'image(i);
        assert (outputs = output_arr1(i))
        report "Output does not match expected at test "
        & integer'image(i)
        severity error;
        -- if output is unexpected give an error and report where that error happened
        end loop;
        report "Test Finished :)";
  end process;
  remote_1: remote_control port map (inputs(0), inputs(1), inputs(2), inputs(3), inputs(4), inputs (5), inputs(6), clk,
                                     outputs(0), outputs(1), outputs(2));
end test1;
```

## 7.3. Further FPGA information

The FPGA used for this project was the Basys 3 Artix-7 FPGA Board [10, 11] as used in the rest of CENG 3010. The parts of the FPGA used to implement the final solution were the switch inputs and LED outputs of the board. The internal clock of the FPGA is 100 MHz. Due to the bounce effect of the buttons, they were not used in the implementation of this project.