**All of the following functions are public functions in the DiscoCubesSM.c module:**

/\*This function initializes the DiscoCubes state machine, setting the current state to “dormant” and ensuring that the direction registers of all pins are set correctly to read and write as appropriate.\*/

**boolean InitDiscoCubesSM(void)**

Takes no parameters, returns no value.

Set CurrentState to be Dormant State

Initialize all used port lines of Microcontroller

Ports 0 1 and 2 are set to PWM in the PWM init function.

Port 0 controls motor 1, Port 1 controls motors 2,3.

Set PTT4 as input for 74HC165 for Magnetic Reed Switches

Set PTT5 as output to L293B for enable

Set PTT6 as input for sensing IR limit switch state (down)

Set PTT7 as input for sensing IR limit switch state (up)

Set PTM0 as output to RCK of shift register for Cube LEDs

Set PTM1 as output to datainput of shift register for Cube LEDs

Port initialization for instruction LED in front panel:

Set Port M2 as output to Datainput of Shift Register for LED panel

Set Port M3 as output to RCK of both 74hc595s for LED panel

Set Port M4 as output to buzzer

Set Port M5 as output to all SCKs.

Initialize Port T, Port M to 0

Port AD Initialization:

AD0 is the knob, AD1 is shift/~load for the 74HC165, AD2 is the start button,

AD3 is NOTHING and AD4 to AD7 are the buzzers.

Initialize AD0 as analog input of Difficulty Level

Initialize Port AD to 0

Initialize PWM capabilities

Call ClearAllLEDs Function

Initialize the event queue for DiscoCubesSM

End of InitDiscoCubesSM

/\*This function runs the DiscoCubes state machine, responding to events depending on the state of the machine. Essentially, the machine can be idling (the dormant state), initializing (after the player hits the start button), running (when the initialization is complete), or in either a win or lose state (depending on whether or not the player solved the puzzle in time).\*/

**boolean RunDiscoCubesSM(void)**

Local Variable ThisEvent

Local Static Variable: difficulty-- difficulty level

Local Static Variable: led=0 -- tells us which front panel LED to light

Local Static Variable: n -- number of times the user puts a wrong cube

Pull an event from the queue

If Event Type is EF\_NO\_EVENT

{

Return False

}

Else

{

Switch (CurrentState)

case Dormant:

Set n (number of mistakes) to 0

If Pillars are in the middle --IR Down Limit Switch is not reached

{

Call StartMotorDown with a speed of 30

}

If Event Type is IR\_DOWN\_LIMIT\_SWITCH --IR Down Limit Switch is reached

{

Call StopAllMotors function

}

If Event Type is BUTTON\_PUSHED -- start button has been pushed

{

Call ClearAllLEDs function

Call InitAnswers function to initialize a random Front Panel LED pattern

Set difficulty to value called from ReadDifficultyKnob function

Set CurrentState to Game\_Initializing which takes us to game initializing state

Call StartMotorUp function to start motor to raise pillars with a set difficulty level; IR sensor has a limit switch to stop

}

If Event Type is EF\_INIT or EF\_TIMEOUT

{

switch the result of led%5 to choose a Front Panel LED

Decalre local int Variable i

case 0:

Set PanelLED to the 0th LED and store all colors

Repeat 16 times

{

if the least significant bit of PanelLED is High

{

Set Port M2, which is connected to the data input for U4&U5 74HC595, to High

}

Else

{

Set Port M2 to Low

Pulse the SCK line of U4&U5 74HC595

Shift PanelLED one bit to the right

}

}

Pulse the RCK line of U4&U5 74HC595

Increase led by 1

Call EF\_Timer\_InitTimer(0, 500) to Initilaize Timer with 500 ms

End of case 0

case 1:

Set PanelLED to the 1st LED and store all colors

Repeat 16 times

{

if the least significant bit of PanelLED is High

{

Set Port M2, which is connected to the data input for U4&U5 74HC595, to High

}else

{

Set Port M2 to Low

Pulse the SCK line of U4&U5 74HC595

Shift PanelLED one bit to the right

}

}

Pulse the RCK line of U4&U5 74HC595

Increase led by 1

Call EF\_Timer\_InitTimer(0, 500) to Initilaize Timer with 500 ms

End of case 1

case 2:

Set PanelLED to the 2nd LED and store all colors

Repeat 16 times

{

if the least significant bit of PanelLED is High

{

Set Port M2, which is connected to the data input for U4&U5 74HC595, to High

}else

{ Set Port M2 to Low

Pulse the SCK line of U4&U5 74HC595

Shift PanelLED one bit to the right

}

}

Pulse the RCK line of U4&U5 74HC595

Increase led by 1

Call EF\_Timer\_InitTimer(0, 500) to Initilaize Timer with 500 ms

End of case 2

case 3:

Set PanelLED to the 3rd LED and store all colors

Repeat 16 times

{

if the least significant bit of PanelLED is High

{ Set Port M2, which is connected to the data input for U4&U5 74HC595, to High

}else

{ Set Port M2 to Low

Pulse the SCK line of U4&U5 74HC595

Shift PanelLED one bit to the right

}

}

Pulse the RCK line of U4&U5 74HC595

Increase led by 1

Call EF\_Timer\_InitTimer(0, 500) to Initilaize Timer with 500 ms

End of case 3

case 4:

Set PanelLED to the 4th LED and store all colors

Repeat 16 times

{

if the least significant bit of PanelLED is High

{

Set Port M2, which is connected to the data input for U4&U5 74HC595, to High

}else

{ Set Port M2 to Low

Pulse the SCK line of U4&U5 74HC595

Shift PanelLED one bit to the right

}

}

Pulse the RCK line of U4&U5 74HC595

Increase led by 1

Call EF\_Timer\_InitTimer(0, 500) to Initilaize Timer with 500 ms

End of case 4

End of Switch result of led%5

case Game\_Initializing :

If Event Type is IR\_UP\_LIMIT\_SWITCH

{

Call StopAllMotors function

Call DisplayAnswer function to light up the LED display on the front panel showing answer

Call TurnOnCubeLEDs function to enable all cube LEDs circuit connections

}Else if Event Type is CUBE\_INCORRECT

{

Call ActivateBuzzer function to activate buzzer under this incorrect cube with returned number of cubes from EventCheckers.c

}Else if Event Type is CUBE\_CORRECT

{ Set CurrentState to Game\_Running State

Call EF\_Timer\_InitTimer(0, 35000) to initialize timer with 35 seconds

Call UpdateUsersAnswer function to update the cube correct/incorrect information

Call StartMotorDown with difficulty level to to lower pillars

}

case Game\_Running :

If Event Type is CUBE\_CORRECT --user has placed a cube correctly during game play

{

Call UpdateUsersAnswer function to update the cube correct information

If the returned value called from CompareAnswers

function is True (the user’s answer and the computer’s answer are the same)

{

Set CurrentState to Game\_Over\_Win State

Post Event: GAME\_OVER

}

}

Else if Event Type is CUBE\_INCORRECT

{

Increase the count of Cube Incorrect number n by 1

Call ActivateBuzzer function to activate buzzer under this incorrect cube with returned number of cubes from EventCheckers.c

If n is larger than or equal to 2 – Cube placed incorrectly two or more times

{

Call Switch2Colors function to switch 2 colors on the Front Panel LED display

}

}Else if Event EventType is IR\_DOWN\_LIMIT\_SWITCH – pillars reach the bottom

{

Call StopAllMotors function

Set CurrentState to Game\_Over\_Lose State

Post Event: GAME\_OVER

}Else if Event Type is EF\_TIMEOUT – time out

{ Call StopAllMotors fuction

Set CurrentState to Game\_Over\_Lose State

Post Event: GAME\_OVER

}

case Game\_Over\_Win :

if event type is GAME\_OVER

{ Call YouWinLEDMessage fucion

Call StartMotorUp function

}

if Event Type is IR\_UP\_LIMIT\_SWITCH

{ Call AlternatePillars function

Call StartMotorDown function

}Else if Event Type is IR\_DOWN\_LIMIT\_SWITCH

{ Call StopAllMotors function

Call ClearAllLEDs function

Set CurrentState to Dormant

Post Event: EF\_INIT

}

case Game\_Over\_Lose :

if Event Type is GAME\_OVER

{

Call ClearAllLEDs function

Set CurrentState to Dormant

Post Event: EF\_INIT;

}

}

return True

}

End of RunDiscoCubesSM

**All of the following functions are private functions to the DiscoCubesSM.c module:**

/\*This function initializes the module level variables computersanswer and usersanswer. Computersanswer stores the randomly generated pattern the player tries to match, while usersanswer stores the answer currently provided by the user.\*/

**returns nothing: InitAnswers (takes nothing)**

{

declare a local variable i of type integer

for variable i going from 0 to 4 (repeat the following 5 times):

{

set element i of computersanswer equal to the result of calling the function RandColor

}

for variable i going from 0 to 5 (repeat the following 5 times):

{

set element i of usersanswer equal to ‘#’ (indicating no answer)

}

}

End of InitAnswers

/\*This function reads the current setting of the analog knob which represents difficulty (motor speed). This number is normalized to an integer between 0 and 50. We require it to be between 0 and 50 (not 0 and 100) because we are directly pulse-width modulating the

input of the L298NE – therefore a duty cycle of 50 represents no motion (equal signals forward and backward).\*/

**returns an integer: ReadDifficultyKnob (takes nothing)**

{

declare a local variable difficulty of type integer

read the analog value on Port AD pin 0 and set the result equal to difficulty

normalize difficulty by setting it equal to (10\*difficulty/1024)\*5

return difficulty

}

End of ReadDifficultyKnob

/\*This function reads the internal representation of the computer’s answer and physically displays this pattern of colors on the LED panel on the front of the box. \*/

**returns nothing: DisplayAnswer (takes nothing)**

{

declare a local variable i of type character

declare a local variable j of type character

declare a local variable cube\_pattern of type integer and set it equal to 0.

for the variable j going from 0 to 4 (repeat the following 5 times):

{

declare a local variable color of type character

execute the following code depending on what color is:

{

case color is 0:

set cube\_pattern to the sum of cube\_pattern and the bitwise representation of “blue” shifted over by j\*3

case color is 1:

set cube\_pattern to the sum of cube\_pattern and the bitwise representation of “green” shifted over by j\*3

case color is 2:

set cube\_pattern to the sum of cube\_pattern and the bitwise representation of “red” shifted over by j\*3

}

}

for the variable i going from 0 to 15 (repeat the following 16 times):

{

if the first bit of cube\_pattern is 1

{

set Port M pin 2 high (this is the data input for the 74HC595).

}else

{

set Port M pin 2 low.

}

pulse the shift clock for both 74HC595 shift registers

shift the binary bitwise representation of cube\_pattern to the right by 1

}

pulse the register clock for both 74HC595 shift registers

}

End of DisplayAnswer

/\*This function updates the internal representation of the user’s current answer in the case that the user correctly places a cube on a pillar.\*/

**returns nothing: UpdateUsersAnswer (takes an integer, param)**

{

set element param of usersanswer equal to element param of computersanswer

}

End of UpdateUsersAnswer

/\*This function generates a random character (0,1 or 2) which correspond to the colors blue, green and red.\*/

**returns a character: RandColor (takes nothing)**

{

declare a local variable randomcolor of type character

seed the random number generator with the current time

set randomcolor equal to the result of generating a random positive integer and taking the modulus with 3

return randomcolor

}

End of RandColor

/\*This function turns on the switches that control current flowing to the cubes resting on top of the pillars, thereby causing all cubes to turn on.\*/

**returns nothing: TurnOnAllCubeLEDs (takes nothing)**

{

declare a local variable i of type integer

for the variable i going from 0 to 4 (repeat the following 5 times):

{

set Port M pin 1 to high (this is the data input of the shift register controlling the cube LEDs)

Pulse the shift clock of the shift register controlling the cube LEDs

}

pulse the register clock of the shift register controlling the cube LEDs

}

End of TurnOnAllCubeLEDs

/\*This function switches two colors at random on the LED display panel and updates the computer’s answer to reflect this switch. This function is called in response to the user placing an incorrect cube on the pillars more than once.\*/

**returns nothing: Switch2Colors (takes nothing)**

{

declare a local variable temp of type character

declare a local variable randomcube2 of type character

declare a local variable randomcube1 of type character

seed the random number generator with the current time

set randomcube1 equal to the result of calling the random

number generating and taking the modulus with 5

set randomcube2 equal to the result of calling the random

number generating and taking the modulus with 5

set temp equal to element randomcube1 of computersanswer

set element randomcube1 of computersanswer equal to element randomcube2 of computersanswer

set element randomcube2 of computersanswer equal to temp

call the function DisplayAnswer

}

End of Switch2Colors

/\*This function starts the motors pushing the pillars upward. N.B. at a duty cycle of 50, the motor will not move, since in the H-bridge configuration, low corresponds to moving in one direction, and high in another. Therefore, we modify "difficulty" to reflect this fact. Furthermore, we can exploit this phenomenon in StartMotorUp and StartMotorDown. Lastly, motor 1 is wired

opposite to motors 2,3 and so a duty cycle of 70 for all motors sets motors 2,3 moving in one direction and motor 1 moving in the opposite direction.\*/

**returns nothing: StartMotorUp (takes an integer, difficulty)**

{

set Port T pin 5 to high (this enables the L293NE)

set the duty cycle on Port T pin 0 to 50+difficulty

set the duty cycle on Port T pin 1 to 50+difficulty

set the frequency of the pulse width modulation to 250 Hertz

}

End of StartMotorUp

/\*This function starts the motors guiding the pillars downward. The difficulty is modified to reflect the fact that gravity helps us move, so a lower duty cycle is required to keep the speed of travel downward close to the speed of travel upward.\*/

**returns nothing: StartMotorDown (takes an integer, difficulty)**

{

set Port T pin 5 to high (this enables the L293NE)

set the duty cycle on Port T pin 0 to 50-difficulty/4

set the duty cycle on Port T pin 1 to 50-difficulty/4

set the frequency of the pulse width modulation to 250 Hertz

}

End of StartMotorDown

/\*This function turns off all LEDs (both those on the front panel and those inside the cubes).\*/

**returns nothing: ClearAllLEDs (takes nothing)**

{

declare a local variable j of type integer

declare a local variable k of type integer

for the variable j going from 0 to 15 (repeat the following 16 times):

{

set Port M pin 2 low (this is the input for the shift registers controlling the front panel LEDs)

Pulse the shifter register clock of the shift registers controlling the front panel LEDs

}

Pulse the register clock of the shift register controlling the

front panel LEDs

for the variable k going from 0 to 4 (repeat the following 5 times):

{

set Port M pin 1 low (this is the input for the shift register controlling the cube LEDs)

Pulse the shifter register clock of the shift register controlling the cube LEDs

}

Pulse the register clock of the shift register controlling the

cube LEDs

}

End of ClearAllLEDs

/\*This function pulses the shift clock of the shift registers that control the front panel LEDs.\*/

**returns nothing: PulseSCKU4U5 (takes nothing)**

{

declare a local variable i of type integer

set Port M pin 5 low

declare a dummy variable to waste some time, ensuring a clean transition from low to high

set Port M pin 5 high

}

End of PulseSCKU4U5

/\*This function pulses the shift clock of the shift register that controls the shift register for the cube LEDs.\*/

**returns nothing: PulseSCKU1 (takes nothing)**

{

declare a local variable i of type integer

set Port M pin 5 low

declare a dummy variable to waste some time, ensuring a clean transition from low to high

set Port M pin 5 high

}

End of PulseSCKU1

/\*This function pulses the register clock of the shift registers that control the front panel LEDs.\*/

**returns nothing: PulseRCKU4U5 (takes nothing)**

{

declare a local variable i of type integer

set Port M pin 3 low

declare a dummy variable to waste some time, ensuring a clean transition from low to high

set Port M pin 3 high

}

End of PulseRCKU4U5

/\*This function pulses the register clock of the shift register that controls the cube LEDs.\*/

**returns nothing: PulseRCKU1 (takes nothing)**

{

declare a local variable i of type integer

set Port M pin 0 low

declare a dummy variable to waste some time, ensuring a clean transition from low to high

set Port M pin 0 high

}

End of PulseRCKU1

/\*This function activates the buzzer under the pillar cap of a cube which is placed incorrectly.\*/

**returns nothing: ActivateBuzzer (takes an integer, param)**

{

declare a local variable t of type integer and set it equal to the current time

based on the value of param, execute one block of the following code

{

case param is 0:

set Port AD pin 4 high (activating the buzzer)

do nothing for 0.5 seconds

set Port AD pin 4 low

case param is 1:

set Port AD pin 5 high (activating the buzzer)

do nothing for 0.5 seconds

set Port AD pin 5 low

case param is 2:

set Port AD pin 6 high (activating the buzzer)

do nothing for 0.5 seconds

set Port AD pin 6 low

case param is 3:

set Port AD pin 7 high (activating the buzzer)

do nothing for 0.5 seconds

set Port AD pin 7 low

case param is 4:

set Port M pin 4 high (activating the buzzer)

do nothing for 0.5 seconds

set Port M pin 4 low

}

}

End of ActivateBuzzer

/\*This function is called when the program determines that user has successfully matched the computer’s answer. Initially, all cubes are on. This function turns each cube off one at a time in order, and then back on, one at a time. This is part of the end-of-game show associated with a winning outcome.\*/

**returns nothing: YouWinLEDMEssage (takes nothing)**

{

declare a local variable i of type character

declare a local variable j of type character

declare a local variable startTime of type integer

declare a local variable cubesoff of type integer and set it equal to the hexadecimal number ~0x1

declare a local variable cubesoff of type integer and set it equal to the hexadecimal number ~0x1

declare a local variable cubeson of type integer and set it equal to the hexadecimal number 0x1

declare a local variable maskon of type integer and set it equal to the hexadecimal number 0x1

for the variable j going from 0 to 4 (repeat 5 times):

{

set cubesoff equal to cubesoff masked with maskoff

for the variable i going from 0 to 4 (repeat 5 times):

{

if cubesoff shifted bitwise to the right i times has a 1 in its least significant digit

{

set Port M pin 1 high

}else

{

set Port M pin 1 low

}

pulse the shift clock of the shift register controlling the cube LEDs

}

pulse the register clock of the shift register controlling the cube LEDs

set startTime equal to the current time

do nothing for 0.5 seconds

set cubesoff equal to the result of shifting cubesoff to the left bitwise by 1

}

for the variable j going from 0 to 4 (repeat 5 times):

{

set cubeson equal to cubeson masked with maskon

for the variable i going from 0 to 4 (repeat 5 times):

{

if cubeson shifted bitwise to the right i times has a 1 in its least significant digit

{

set Port M pin 1 high

}else

{

set Port M pin 1 low

}

pulse the shift clock of the shift register controlling the cube LEDs

}

pulse the register clock of the shift register controlling the cube LEDs

set startTime equal to the current time

do nothing for 0.5 seconds

set cubesoff equal to the result of shifting cubesoff to the left bitwise by 1

}

}

End of YouWinLEDMessage

/\*This function sends all pillars down for a little, then sends the central pillar up and the outer pillars down in an alternating fashion as part of the end-of-game show associated with a winning outcome.

**returns nothing: AlternatePillars (takes nothing)**

{

declare a local variable startTime of type integer

declare a local variable i of type integer

set Port T pin 5 high to enable the L293NE

set Port T pin 3 low to turn on motor 1 in the down direction

set Port T pin 4 low to turn on motors 2 and 3 in the down direction

set startTime equal to the current time

wait for 1 second

set Port T pin 3 high to now split the central pillar from the outside pillars, sending it upwards

set Port T pin 4 low keeping the outer pillars moving downwards

set startTime equal to the current time

wait for 1 second

for the variable i going from 0 to 1 (repeate twice)

{

set Port T pin 3 low to send the central pillar back down

set Port T pin 4 high to send the outer pillars back up

set startTime equal to the current time

wait for 2 seconds

set Port T pin 3 high to send the central pillar back up

set Port T pin 4 low to send the outer pillars back down

set startTime equal to the current time

wait for 2 seconds

}

set Port T pin 3 low sending the central pillar down

set Port T pin 3 high sending the outer pillars up

set startTime equal to the current time

wait for 1 second, bringing us back to where we were at the start

}

End of AlternatePillars

/\*This function turns off all motors.\*/

**returns nothing: StopAllMotors (takes nothing)**

{

set Port T pin 5 low to disable the L293NE

}

End of StopAllMotors

/\*This function compares the internal representation of thcomputer’s answer with the answer currently provided by the user to determine whether or not the user has completed the puzzle.\*/

**returns True or False: CompareAnswers (takes nothing)**

{

declare a local variable i of type integer

for the variable i going from 0 to 4 (repeat 5 times):

{

if element i of usersanswer does not equal element i of computersanswer

{

return False

}

}

return True (if we made it here, that means that every element had to match)

}

End of CompareAnswers