A real car!

Yan got his driver's license, and really want to have a high-tech real car. One day, Yan bought a toy car, and it is really easy to use. It only has 4 operations: go straight, go back, turn left, turn right. ... "Hey, wait. My car can't be this crap."

To experience what it's like to drive a real car, Yan want to change the way the car operates, and he ask you for help. Therefore, you connect it to your EGO1 developing board. You need to develop a program that translate the "complex operation" of a real car into the 4 simple operations of a small toy car.

Also, since Yan want a high-tech real car, so you are expected to develop manual driving mode, semi-auto driving mode and auto driving mode.

Although it is only a toy car, it carries Yan's fantasy dream. Can you help him?

[TOC]

Basic: Global State (20%)

The car has 2 states: **power-on** and **power-off**. We would use 2 buttons to manage the basic status switching.

Power on (button)

Press and hold on power on button for at least 1 seconds, the car will enable its engine.

If the car's engine is not enabled, all the other input is unavailable.

Power off (button)

Press the power off button, the car will disable its engine. Later in the introduction, there are some special situations that can cause the engine disabled.

When the car is power-on, it has at most 3 modes that can be selected:

Manual driving mode, Semi-auto driving mode, Auto driving mode(alternative)

Then we would introduce this three mode seperately.

Basic: Manual Driving (50%)

In manual driving mode, the car can be operated as a real car.

State Analysis

In Manual driving mode, the car has 3 states:

not-starting, starting, moving.

Initially (When the car is enable its engine, or switch to manual mode), the car should in not-starting state. The car can move if and only if it is in moving state.

Input

Throttle (switch)

If the car is started, you can directly use throttle to control whether the car should move.

But if the car is not started, you need throttle with clutch t start the car, otherwise the car may lose its power!

```
When turning on the throttle:
If the car is in starting state, switch to moving state.
If the car is in not-starting state, we should check the clutch.

When turning off the throttle:
If the car is in moving state, switch to starting state.
```

Clutch (switch)

Whenever the clutch is enabled, the car cannot move. In the case that the car is not started, We need clutch to get the car start. We should firstly keep the clutch on, then turn on the throttle to complete the car start. After doing this, we finish starting.

If we directly turn on the throttle (without turning on the clutch) when the car need to start, the car engine will stop working and need to power on again.

```
When the car is in not-starting state:
turn on the throttle **with clutch enabled**, switch to starting state.
turn on the throttle **without clutch enabled**, the car would power off.

When the car is in starting/moving state:
if clutch is on, the car must leave the moving state, and can not switch to moving
state.
```

Brake (switch)

Turn on brake switch to let the car stop. After braking, the car need to start again.

Motice that most of cars have the Brake override system (BOS): If we turn on brake and throttle at the same time, brake should work and throttle should not work. Yan's car should also have BOS.

```
When the car is in starting/moving state:
turn on the brake, switch to not starting state.

If the brake is on, throttle switch is invalid.
```

Reverse gear shift (switch)

If this switch is on, the car should move back instead of move forward.

When switching this switch, the clutch must keep turning on, otherwise the car would lose power.

```
When the car is in moving state:
If the reverse gear shift is on, the car should move back.
If the reverse gear shift is off, the car should move forward.
If we switch the reverse gear shift in moving state without clutch enabled, the
car would power off.
```

Turn left and turn right (button)

```
If the car is in starting state or moving state:
When only turn left button pressed, the car should turn left.
When only turn right button pressed, the car should turn right.
Otherwise (no one is pressed or both are pressed), the car should go straight.
```

Output

Car Operation (to UART)

You can control the car by activating the signals in uart module. There are 4 signals in total: go straight, go back, turn left, turn right.

For more details, please go to UART Module introduction to check it out.

Turn Light (LED)

There are two LED that act as turn light: If the car is not starting, both of the LED should keep on. Else if the car is turning left, the LED on the left should be flashing.

Else if the car is turning right, the LED on the right should be flashing.

Mileage record (7 seg)

When the car is power on, the mileage record needs to be displayed in seven digital tubes. The mileage should keep increasing when the car is in moving state.

When the car is power off, the mileage should reset to 0 and no values are displayed (maybe "-" or just nothing displayed).

Advanced: Semi-Auto Driving (30%)

In semi-auto driving mode, the car will automatically go straight, stop at a fork in the road, then wait for our command (straight or turn left or turn right).

State Analysis

The car has 3~4 states in semi-auto driving mode: Moving, Turning(turn-left and turn-right), Wait-for-command

Initially, the car is in moving state. When the car get to a fork in the road, it would switch to wait-for-command state. After sending a command, the car would switch to turning state or moving state according to commands.

Input

Detector (from UART)

There are 4 detector in each direction of the car. you can use them to detect whether there are barriers near the car.

Semi-auto driving command (multi button)

The car would automatically go straight. We can use button to let the car know whether it should go straight, turn left, turn right or go back in a fork.

```
In moving/turning state:
The command button is invalid.

In wait-for-command state:
3 buttons can switch into 3 corresponding states: turn-left, turn-right, go straight.
Turning states would automatically switch into moving states when the car turns 90 degrees.
```

Hint

1. You should use the informations of 4 detectors to determine if the car is at a fork in the road.
2. In turning state, you need to estimate the time to turn 90 degrees. Turning states can only be automatically switched accorting to **time**. Provide a 50Hz clk is a good choice.
3. The value of 4 detector may not change at the same time(time difference of about 40ms). Do not let detector be your trigger signal in "always"!
4. It is recommended to add a cooldown between turning and going straight. Because your car takes a short time to automatically calibrate the orientation.

Bonus: Auto Driving (extra 20%)

If the car is in auto driving mode, then the car should automatically drive out a maze with no extra operation!

Input

Place beacon (to UART)

In the posedge of this signal, the car would place a beacon in place. The beacon would also be detected by your car's detector (like a barrier), but the car can go through the beacon.

Up to 10 beacons can exist in the map at the same time. If there are more than 10 beacons, the first one placed will be destroyed. (WARNING: this signal must keep active for more than 20ms, otherwise it will be ineffective because of the buffeting mechanism.)

Destroy beacon (to UART)

In the posedge of this signal, the car would destroy the most recently placed beacon. (WARNING: this signal must keep active for more than 20ms, otherwise it will be ineffective because of the buffeting mechanism.)

Hint

1. Right turn first principle is helpful to get out of a maze, but be careful: A loop in the maze can invalidate this principle.
2. The maze we prepared for testing auto-driving is simple enough. Your car is required to reach destination within a limited time.
3. Beacon is very helpful for detecting loop and traversing mazes.
4. Even if your car doesn't make it to the destination, you can get guaranteed bonus points as long as it's fully autonomous and do not hit the wall.

Bonus: VGA (extra 20%)

Task1. VGA available

Use switch to control VGA to display any thing.

Task2. use VGA to show the state

Let VGA show the state of your car.

Task3. use VGA to show the Mileage record

Let VGA show the Mileage record. (real-time synchronization)

Appendix: Introduction to UART module

The module declaration is as follows:

```
module SimulatedDevice(
    input sys_clk, //system clock (100Hz, P17 pin)
    input rx, //bind to N5 pin
    output tx, //bind to T4 pin

    input turn_left_signal,
    input turn_right_signal,
    input move_forward_signal,
    input move_backward_signal,
    input place_barrier_signal,
    input destroy_barrier_signal,
    output front_detector,
    output back_detector,
    output left_detector,
    output right_detector
    );
```

system interface

sys_clk,rx,tx requires direct access to the pin:

```
sys_clk ---- P17
rx      ---- N5
tx      ---- T4
```

input interface

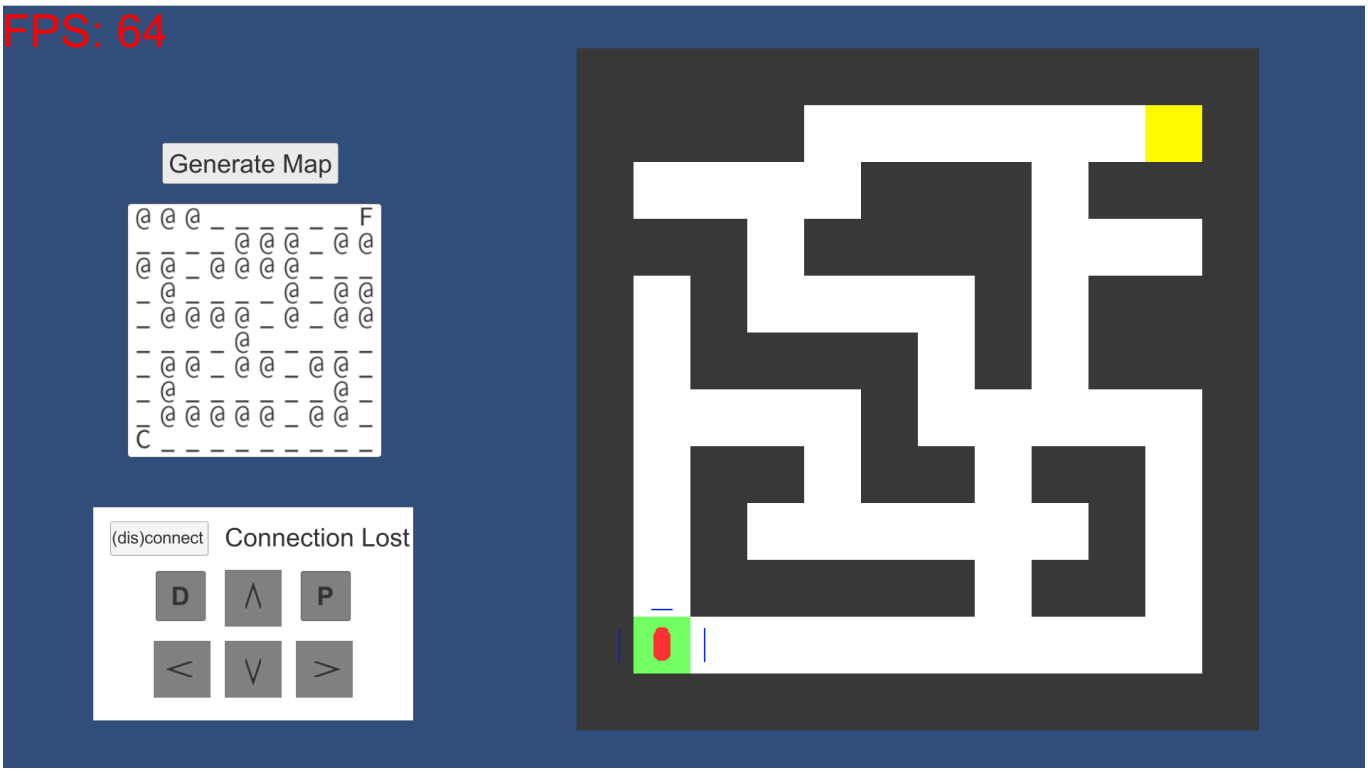
The 6 input signal allows you operate the car: **turn_left_signal** the car would keep turning left if this signal is active. **turn_right_signal** the car would keep turning right if this signal is active. **move_forward_signal** the car would keep moving forward if this signal is active. **move_backward_signal** the car would keep moving backward if this signal is active.

place_barrier_signal the car would place a beacon in the posedge of this signal. (WARNING: this signal must keep active for more than 20ms, otherwise it will be ineffective because of the buffeting mechanism) **destroy_barrier_signal** the car would destroy the most currently placed beacon if this signal is active. (WARNING: this signal must keep active for more than 20ms, otherwise it will be ineffective because of the buffeting mechanism)

output interface

The 4 output signal is the information from the detector. **front_detector** If the detector in the front of the car find a wall/becon (In other words, there are walls in the front of the car), this signal would be active. **back_detector** If there are walls in the back, this signal would be active. **left_detector** If there are walls on the left, this signal would be active. **right_detector** If there are walls on the right, this signal would be active.

Appendix: Simulation Software Introduction



The whole software has three parts: map generator, controller, the maze.

Map Generator

We use 4 characters represent 4 types of objects in the maze: **@** is the wall in the maze. _ is the road in the maze. **C** is the car in the maze. **F** is the destination in the maze.

Controller

It is responsible for connecting the development board through the UART and receiving control signals. If the controller get control signals from your develop board, you can see the corresponding button highlighted.

The maze

You can see your car (red) in the maze. Your goal is driving the car to the destination (yellow).

You may see four blue lines around the car, they are the effective operating range of the detector.

Appendix: How to start?

Resources list

We have 6 resources:

1. **CarSimulation** The software that provides a simulated car and a maze.
2. **SimulatedDevice_demo.bit** This is the demo of Simulated Device that can run directly.
3. **SimulatedDevice_src** The source code of the demo.

4. **State_reference_diagram.pdf** State transition for reference. For reference only!!
5. **DL2022F_Project_Introduction.pdf** is just this file.
6. **semi_auto_driving_mode_demo.bit** the demo of semi-auto driving mode.

STEP1. try the SimulatedDevice demo

To use this demo, program **SimulatedDevice_demo.bit** into your develop board, and then run the CarSimulation software. If it is connected successfully, then you can use the first six switches from left to right to control the car. Also, four LED lights counting from right to left shows feedback from the detector! If it connect failed, please check whether the development board is connected and programmed properly, then click the button in the software to reconnect.

STEP2. import SimulatedDevice into your project

All the source code in demo (folder **SimulatedDevice_src**) makes up the devices that interact with the car. The major part of this device is an UART module. You just need to add all the source code file into your own project.

The module you need to instantiate in your project is called **SimulatedDevice** in file **dev_top.v**, and we have just introduced the device with UART module before.

DO NOT modify any code in the demo, and make sure **SimulatedDevice** was only instantiated once.

STEP3. Designing your own module

After finish the first two steps, you may start designing and implementing your module now.
State_referece_diagram helps you to understand the relationships between states, but remember that this diagram is only for reference, not requirement!!

STEP4. trying the Semi-Auto Driving demo

Since the introduction of semi-auto driving mode may not detailed enough, we also provide a demo for semi-auto driving mode. To use the demo, firstly program **Semi_auto_driving_mode_demo.bit**, then run the CarSimulation software.

You can use the first 4 switches from left to right to control the car:

- **P5 pin** go-straight command.
- **P4 pin** turn-left command.
- **P3 pin** turn-right command.
- **P2 pin** turn-back command.