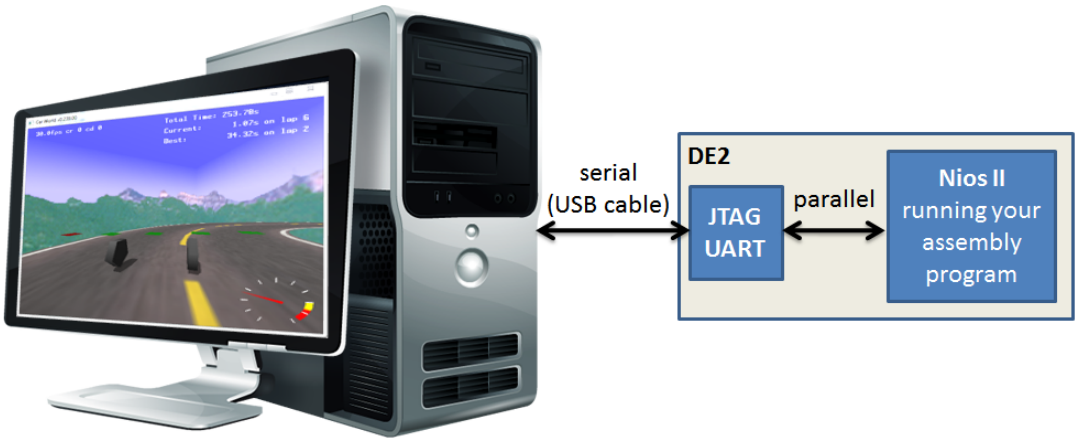
Lab 5: Polled I/O

Description

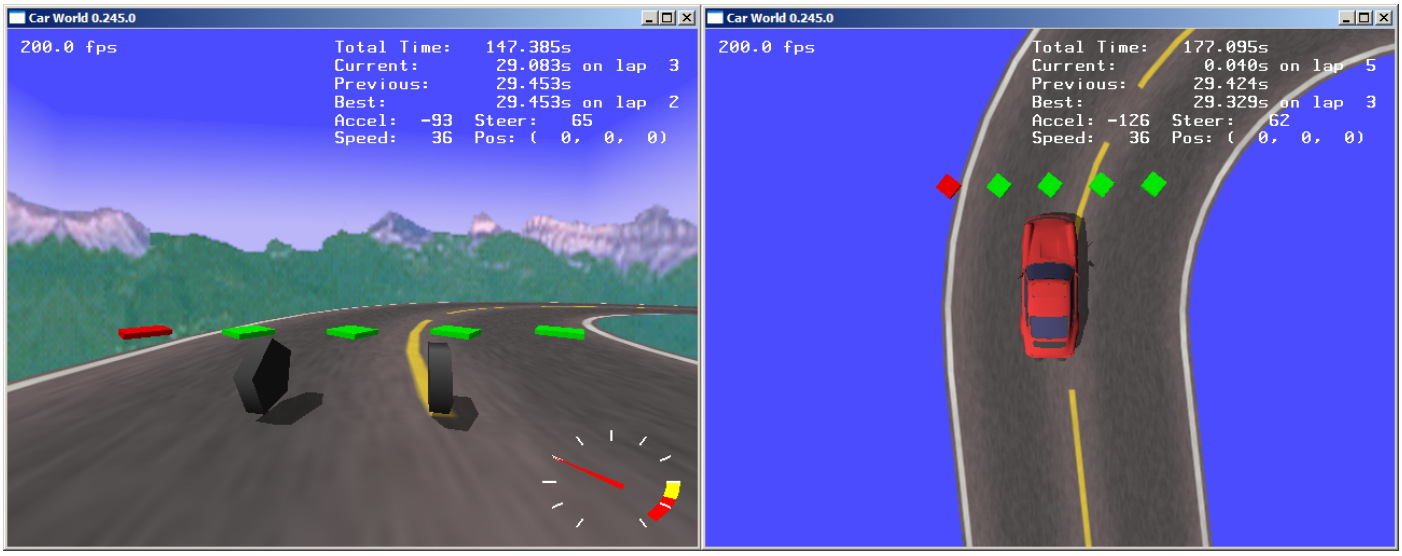
In this lab you will use the *polling* concept for a *JTAG UART* (Universal Asynchronous Receiver/Transmitter that sends data over the JTAG programming cable) to control a simple game. To poll a device is to actively and continuously check its status to see if it is ready to be used in some way.

We will provide the *Car World* game running on a PC. In this game, a car is driven around a race track in order to finish laps as fast as possible without driving off the track. **Your task is to write a Nios II assembly program that will control the car** (without human intervention). The Nios II program you write will communicate with the Car World game over the JTAG UART I/O device. Your program can query the state of the sensors, speed of the car, and current position on the map. Your program must then send acceleration and steering commands to drive the car.



The Car World Game

The goal of the Car World game is to guide the car around the track, trying to complete laps as quickly as possible. If the car drives off the track (or F3), the car will reset to the starting point of the track. Below is a screenshot of the default view of the Car World game (on the left), as well as a screenshot of an alternative view (on the right), accessible through pressing F2.



The car has five sensors in front. These sensors indicate whether the car is approaching the edge of track. For example, on the screenshots above, the leftmost sensor (coloured red) is reporting that it is currently off the track, while the other four sensors (coloured green) are on the track. Your assembly program will need to read the state of these

sensors in order to make an informed decision on moving the car. You can also read the **current speed** of the car and its **current position** (x, y, and z coordinates).

To move the car you need to inform the Car World game of the desired **steering** and the **acceleration/deceleration**. The text in the Car World window reports the current status of the car to help debug your program. To summarize, you are to write a Nios II assembly program that will continuously:

1. Poll the sensors, speed and/or position data from the game,
2. Process the collected data
3. Adjust the steering and acceleration to keep your car moving at a reasonable speed without driving it off the track.

Your assembly program only needs to be able to drive the car so it eventually completes a lap. A reasonable basic algorithm might be to maintain a target speed and steering whenever sensors indicate you are near the edge of the track. Notice that this simple approach does not need to use the position data (x, y, and z coordinates).

However, bragging rights will be given to the group with the best time. Try to optimize your code and have some fun while doing it!

The JTAG UART

Communication between the Car World game and Nios II will use the JTAG UART I/O device. A UART allows a bidirectional stream of bytes to be transferred across some medium, in our case, the JTAG programming cable. A byte you send to the UART will be received by the game on the PC, while a byte sent by the game will be available for you to read.

The documentation of the JTAG UART can be found here.

- **Receiving:** Data Register bit 15 is a "data valid" bit. Poll this bit to check for available bytes to read. If a byte is available, it is available at bits [7:0] and removed from the queue.
- **Sending:** You must ensure the write FIFO queue has enough space to avoid overflowing it. Control register[31:16] tells you the number of free bytes in the write FIFO.

Communicating with the Car World Program

The JTAG UART documentation describes how to receive or send bytes of data. Now we proceed on describing the communication protocol for the Car World program that you will use to communicate with the game.

The communication protocol for the Car World program consists of reading and writing *packets* of data over the JTAG UART. A packet that consists of multiple bytes must be read/written from/to the JTAG UART **one byte at a time**, and you must implement polling for each packet byte. The first byte of every packet is always the packet type. Packet types range from 0 to 5 and are in the details described below. The packet length depends on the type of packet being sent.

Transmit Packet Types

There are in total 4 transmit packet types. These packets are sent by your program running on the Nios II to the Car World game.

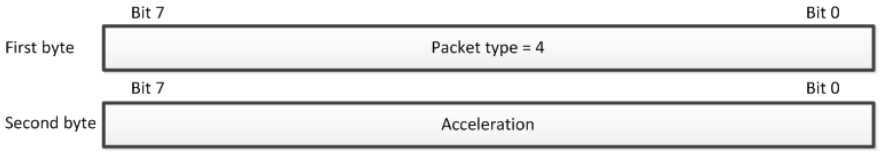
- **Request sensor and speed data**

  To get the current state of the sensors and car's speed, send a one-byte command with value `0x02`. The Car World game will respond by sending you a sensor and speed data packet (type 0), described below.

- **Request position data**

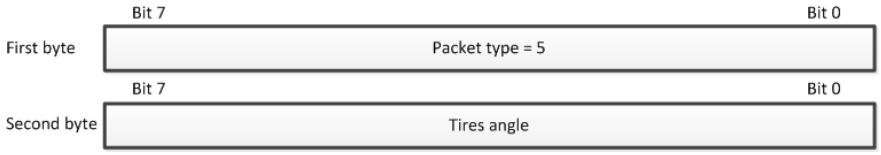
  In order to get car's position, send a one-byte command with value `0x03`. The Car World game will respond by sending the position data packet (type 1).

- **Change acceleration**



To change the acceleration of the car, send a two-byte command to Car World. The first byte (command type) is `0x04`, while the second byte should be the desired acceleration formatted as a 8-bit signed number. The acceleration ranges from -127 (full deceleration) to +127 (full acceleration ahead). It is important to note that an acceleration of 0 does not mean that the car will not be moving; rather, the car will continue to move at its current speed. If the speed reaches zero and you continue to apply a negative acceleration, the car will start moving in reverse. The Car World game sends no response on receiving a packet of this type.

- **Change steering**

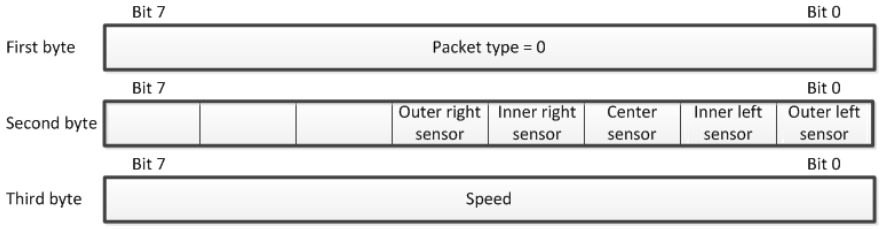


Similarly, changing steering uses a two-byte command. The first byte (command type) is `0x05`, while the second byte should be the desired steering amount formatted as a 8-bit signed number. The steering amount ranges from -127 (hard left turn) through 0 (straight ahead) to +127 (hard right turn). The Car World game sends no response on receiving a packet of this type.

Received Packet Types

There are 2 packet types that are sent by the Car World game as a response to transmit packet types described above.

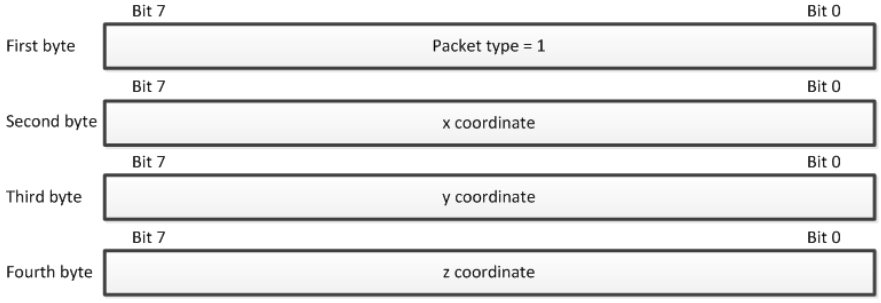
- **Sensor and speed data**



On receiving a packet of type `0x02`, the Car World game sends 3 bytes to the JTAG UART. The first byte is `0x00`, while the other two bytes represent the current value of sensors and the current speed of the car, respectively. The layout of sensor data is shown above. "On the track" sensor state is indicated by a value of 1 and "off the track" state is indicated by a value of 0. For example, the value of the sensor data in the screenshots shown earlier is `0x1e`.

The speed is an 8-bit unsigned number, ranging from 0 (stopped) to 255 (full speed). Note that this is not the velocity of the car, as it only describes how fast the car is moving and not in what direction the car is moving.

- **Position data**



A position data packet consists of 4 bytes, sent by the Car World game as a response to a packet of type `0x03`. The first byte is `0x01`, while the remaining 3 bytes are 8-bit signed numbers representing x,y, and z coordinates, respectively. You are not required to use position data but they may be useful for completing the more difficult tracks.

Pseudocode example

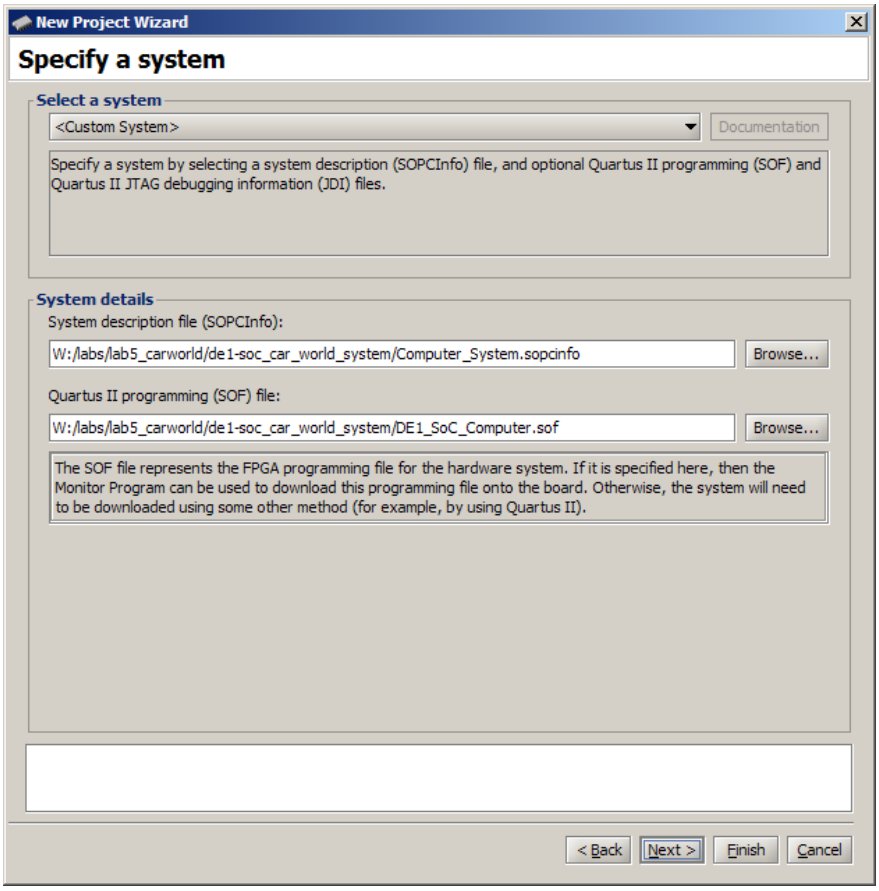
Below is a snippet of pseudo-code showing two subroutines for reading sensor and speed data and adjusting steering.

```
OnePossibleAlgorithm:
  call ReadSensorsAndSpeed()

  # Decide what to do
  if sensors are 0x1f
```

```
    call SetSteering to steer straight
  else if sensors are 0x1e
    call SetSteering to steer right
  else if sensors are 0x1c
    call SetSteering to steer hard right
  else if sensors are 0x0f
    call SetSteering to steer left
  else if sensors are 0x07
    call SetSteering to steer hard left
  else
    Hope this doesn't happen

  # Also do something about the speed.

  ...

ReadSensorsAndSpeed:
  # Request sensors and speed: Send a 0x02.
  call WriteOneByteToUART(2)

  # Read the response
  call ReadOneByteFromUART()
  check that data read is 0
  call ReadOneByteFromUART()
  look at sensor states
  call ReadOneByteFromUART()
  look at current speed
  return

SetSteering:
  call WriteOneByteToUART(5)
  call WriteOneByteToUART(new_steering_value)
  return
```

Lab Setup

1. Download the Lab 5 Car World package and unpack it. To start the game, ensure the DE1-SoC is plugged in, powered on, and programming cable drivers loaded, then run `car_world.bat`. This will:
    1. Download a modified DE1-SoC Computer system onto the FPGA board. (Needed for the non-standard JTAG UART used here)
    2. Start the Car World game and establish the connection with the JTAG UART on the DE1-SoC board.
2. Create a new Altera Monitor project. When selecting a system, choose `<Custom System>`, and specify the `Computer_System.sopcinfo` and `DE1_SoC_Computer.sof` files provided in the directory `lab5_carworld/de1-soc_car_world_system` directory, as shown in the figure below. Leave all the other fields blank.



For program type, specify the `Assembly Program` type, and add the assembly file you wrote that will control the Car World game.

When asked whether you wish to download the computer system onto the FPGA board, choose No. The `car_world.bat` batch file already does this for you.

To do these steps outside of DESL labs, besides having Altera Quartus and Altera Monitor program installed, you will need the DE1-SoC board attached to your computer via a USB cable. **If you don't have the board at home, you still must write the required Nios II assembly code and ensure that it at least compiles.** For this, skip the first step and just compile your code after completing step 2.

You may also test your interactions with the game using the Nios II simulator. It won't talk to the game, but it will respond to the acceleration, steering, and get sensors/speed/position commands, so you can at least test the basic interactions with the game.

Links

- JTAG UART documentation
- Car World computer system and PC game

Preparation and Quiz (3 marks)

1. Read the documentation on the JTAG UART.
2. Write answers to the following questions:
    1. What is the meaning of a sensor reading of `0x00`?
    2. You have sent the Car World game two bytes: `0x05` followed by `0x9C`. What is the meaning of this information?
    3. What bytes would you write to the JTAG UART in order to apply maximum acceleration?
3. Write the assembly code that will guide the car around a track. Don't forget to comment your program, and use functions to keep your code organized!

Be prepared to explain

- Memory-mapped devices
- How to use the JTAG UART
- How polling works

In Lab (2 marks + 1 bonus mark)

Demonstrate your working game. In the directory `lab5_carworld/carworld/data/Landscape` there are multiple tracks labelled for various difficulty levels (easy, medium, hard, and extreme). To change a track:

1. If you are currently running your assembly code in the Altera Monitor Program, select `Actions/Disconnect`.
2. Close all the windows opened by the `car_world.bat`
3. Go to directory `lab5_carworld/carworld/data/Landscape` and copy the desired track file over the file `landscape.txt`. For example, copying `landscape_medium_hill_valley.txt` over `landscape.txt` will switch to the medium track.
4. Run `car_world.bat` again and select `Action/Load` option in the Altera Monitor program.

You will receive:

- 1.5 mark for completing a lap on any one of the easy tracks
- 0.5 mark for completing a lap on the medium track
- 1 bonus mark for completing a lap on the `landscape_hard_hill_valley` track **under 46 seconds**.

Your TA will record your best lap time on `landscape_hard_hill_valley`. Don't forget to submit your code.

For submission, put all of your code in a single assembly file called **lab5.s** and submit it on blackboard.

Notes

- Read and write byte and their associated polling loops are used frequently. It is a good idea to write these as subroutines to reduce duplicated code.
- Car World game commands: F2 to cycle through different viewpoints, F3 to reset the car to the starting point of the track.
- While debugging, you may want to occasionally reset the communication between the Car World game and your assembly program (that is, drain both the receive and send JTAG UART buffers). Doing this takes two steps:
    - Have your assembly code send byte `0x00` to the JTAG UART in case Car World is waiting for a multi-byte packet. Car World treats excess `0x00` commands as a no-op.
    - In the Altera Monitor program, select the `Memory` tab, go to the memory address `0x10001020`, check the `Query Memory Mapped Devices` check-box and click the `Refresh` button. Then **uncheck** the check-box. This is equivalent to reading the UART receive FIFO data until the FIFO is empty.
    - It may be convenient to write a reset subroutine and run it at the beginning of your program.