**Custom Database**
Deadline: 17:00 Monday 17th of November, 2023

**Collaborating in small groups of up to three students is permitted, but you must implement your own programs (absolutely *do not* copy and paste from others) and provide your own answers where appropriate.**

**Note that lacking proper comments will lose mark.**

# Description

We have used combinatorial logic to construct many of the various logic circuits that form the basis of a computer in previous exercises. In this exercise, you will need to implement a database with some simple functionalities. In addition, you will need to incorporate arithmetic operations and overflow detection capabilities in your design. The inputs and outputs of the database can be found in Fig.1.

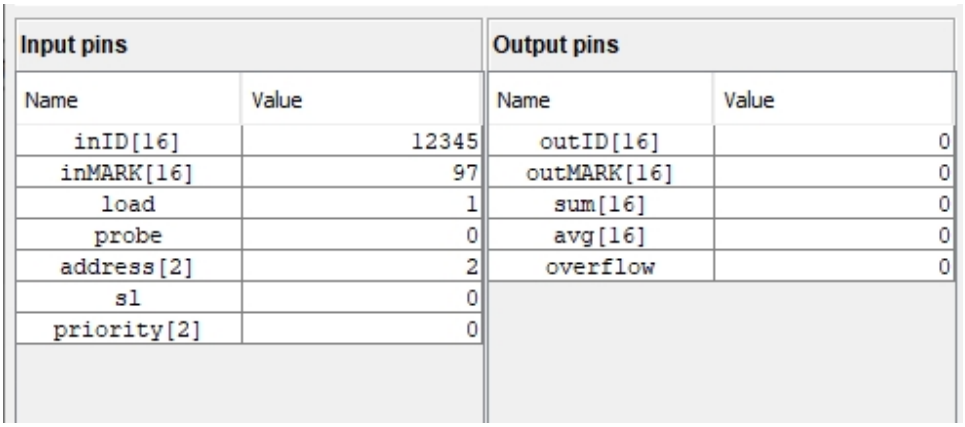
| Input pins | | Output pins | |
|---|---|---|---|
| Name | Value | Name | Value |
| inID[16] | 0 | outID[16] | 0 |
| inMARK[16] | 0 | outMARK[16] | 0 |
| load | 0 | sum[16] | 0 |
| probe | 0 | avg[16] | 0 |
| address[2] | 0 | overflow | 0 |
| sl | 0 | | |
| priority[2] | 0 | | |

Figure 1: Inputs and outputs of the database.

# 1 Task 1: Load and Probe

Load: turn on **Load** (load = 1) to start loading ID (**inID**) and mark (**inMARK**) onto registers assigned by a 2-bit address (**address**). Specifically, you need to implement 4 registers for storing ID and 4 registers for storing mark.

For example, **load** = 1, **inID** = 12345, **inMARK** = 97, **address** = 2. After clicking `simple step` button, the ID and the MARK should be stored in each of 1 of 4 designated registers, as shown in Fig.2.



| Input pins | | Output pins | |
|---|---|---|---|
| Name | Value | Name | Value |
| inID[16] | 12345 | outID[16] | 0 |
| inMARK[16] | 97 | outMARK[16] | 0 |
| load | 1 | sum[16] | 0 |
| probe | 0 | avg[16] | 0 |
| address[2] | 2 | overflow | 0 |
| sl | 0 | | |
| priority[2] | 0 | | |

Figure 2: Example: Load

Functionalities for Load:
**Input**:

1. A 16-bit input ID number called **inID**.
2. A 16-bit input associated mark called **inMARK**.
3. A 1-bit switch for loading the data called **load**.
4. A 2-bit value assigned for **address**.

**No output**.

Next, turn on **probe** (probe = 1) to retrieve data with the assigned address (**address**). For example, after loading the above inputs, when you set **probe** = 1, and **address** = 2, it should output the related ID and mark (**outID** = 12345, **outMARK** = 97) (see Fig.3).

| Input pins | | Output pins | |
|---|---|---|---|
| Name | Value | Name | Value |
| inID[16] | 0 | outID[16] | 12345 |
| inMARK[16] | 0 | outMARK[16] | 97 |
| load | 0 | sum[16] | 0 |
| probe | 1 | avg[16] | 0 |
| address[2] | 2 | overflow | 0 |
| sl | 0 | | |
| priority[2] | 0 | | |

Figure 3: Example: Probe

Functionalities for Probe:
**Input**:

1. A 2-bit value assigned for **address**.
2. A 1-bit switch for probing the data called **probe**.

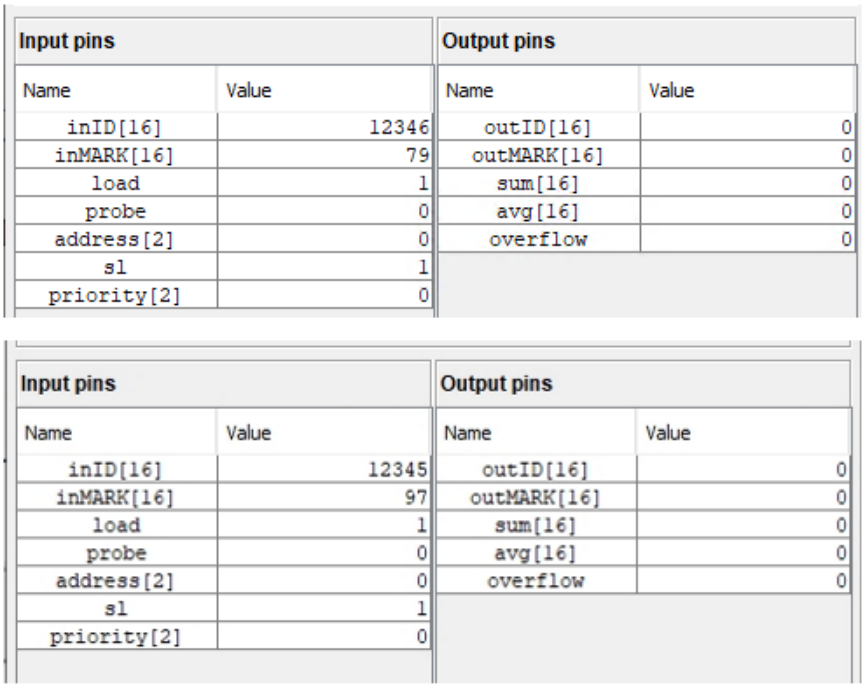
**Output**:

1. A 16-bit output ID number called **outID**.
2. A 16-bit output associated mark called **outMARK**.

(12 marks)

## 2 Task 2: Sequential Load

Based on `Task 1`, implement sequential load functionality. If sequential load (**sl**) is turned on, data will be loaded onto the RAM in the sequential order and overwrite existing values once all the registers (4 registers) are full. Consequently, this creates Sequential Access Memory (SAM). For example, as shown in Fig.4: set **load** = 1 and **sl** = 1, first load student with ID 12346 and mark 79 (stored in the first register by default), then load student with ID 12345 and mark 97 (stored in the second register for the sequential loading). (Generally we assume the inIDs are unique.)



**Input pins**

| Name | Value |
|---|---|
| inID[16] | 12346 |
| inMARK[16] | 79 |
| load | 1 |
| probe | 0 |
| address[2] | 0 |
| sl | 1 |
| priority[2] | 0 |

**Output pins**

| Name | Value |
|---|---|
| outID[16] | 0 |
| outMARK[16] | 0 |
| sum[16] | 0 |
| avg[16] | 0 |
| overflow | 0 |

**Input pins**

| Name | Value |
|---|---|
| inID[16] | 12345 |
| inMARK[16] | 97 |
| load | 1 |
| probe | 0 |
| address[2] | 0 |
| sl | 1 |
| priority[2] | 0 |

**Output pins**

| Name | Value |
|---|---|
| outID[16] | 0 |
| outMARK[16] | 0 |
| sum[16] | 0 |
| avg[16] | 0 |
| overflow | 0 |

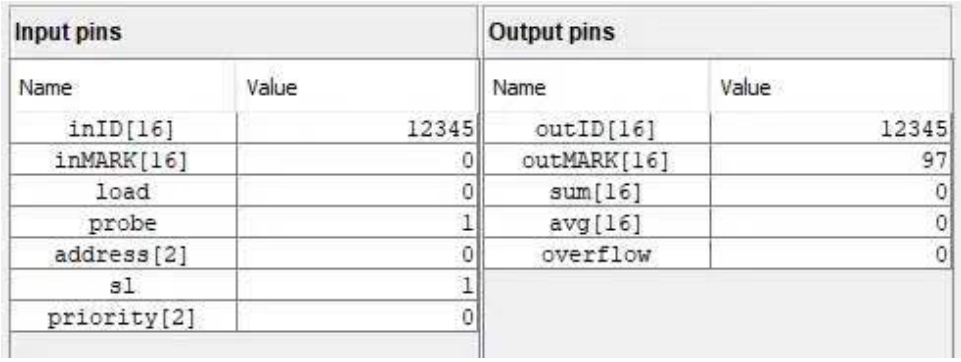
Figure 4: Example: Sequential Load

Functionalities for Sequential Load:
**Input**:

1. A 16-bit input ID number called **inID**.
2. A 16-bit input associated mark called **inMARK**.
3. A 1-bit switch for loading the data called **load**.
4. A 1-bit switch for sequential loading the data called **sl**.

**No output**.

Similarly, turn on **Probe** and **sl** to retrieve the mark with the with assigned **inID**. For example, as shown in Fig.5: set **probe** = 1 and **sl** = 1, with the data already entered above, request the mark for the student with **inID** = 12345. The **outMARK** should show 97 that is related to the same student.



| Input pins | | Output pins | |
|---|---|---|---|
| Name | Value | Name | Value |
| inID[16] | 12345 | outID[16] | 12345 |
| inMARK[16] | 0 | outMARK[16] | 97 |
| load | 0 | sum[16] | 0 |
| probe | 1 | avg[16] | 0 |
| address[2] | 0 | overflow | 0 |
| sl | 1 | | |
| priority[2] | 0 | | |

Figure 5: Example: Sequential Probe.

Functionalities for Sequential Probe:
**Input**:

1. A 16-bit input ID number called **inID**.
2. A 1-bit switch for sequential loading the data called **sl**.
3. A 1-bit switch for probing the data called **probe**.

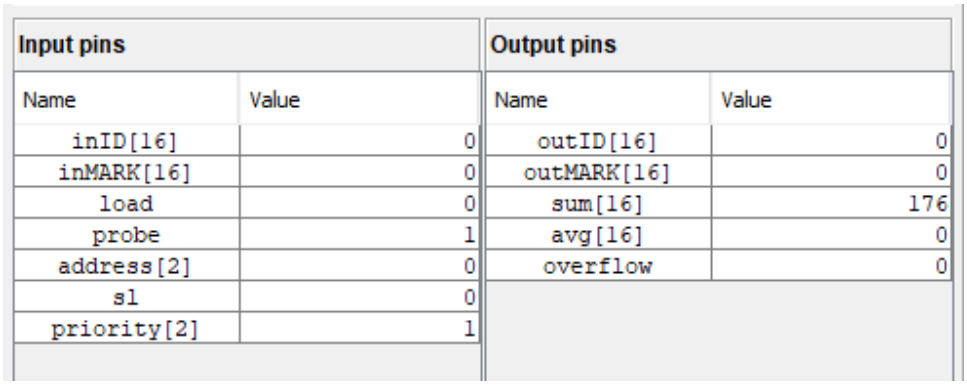
**Output**:

1. A 16-bit output ID number called **outID**.
2. A 16-bit output associated mark called **outMARK**.

(8 marks)

# 3 Task 3: Sum of the Mark

Based on `Task 1` and `Task 2`, use a 2-bit entry called **priority**. When priority equals 1, the chip will calculate the sum of all MARK values stored in the RAM chips and output the value in **sum[16]**. If this value is greater than the maximum value for a 16 bit signed integer using 2's complement, then the overflow flag (**overflow**) should set to 1, otherwise the overflow flag (**overflow**) is 0. Turn on **Probe** and set **priority** = 1 to get the sum of marks. For example, as shown in Fig.6, with the same data loaded in `Task 2`, the sum of the marks (79+97) is 176. Since no overflow occurs, the overflow flag (**overflow**) is 0.



| Input pins | | Output pins | |
|---|---|---|---|
| Name | Value | Name | Value |
| inID[16] | 0 | outID[16] | 0 |
| inMARK[16] | 0 | outMARK[16] | 0 |
| load | 0 | sum[16] | 176 |
| probe | 1 | avg[16] | 0 |
| address[2] | 0 | overflow | 0 |
| sl | 0 | | |
| priority[2] | 1 | | |

Figure 6: Example: Sum

**Input**:

1. A 2-bit switch for function priority called **priority**.
2. A 1-bit switch for probing the data called **probe**.

**Output**:

1. A 16-bit version of the sum of the marks called **sum**.
2. A 1-bit overflow flag called **overflow**.

(5 marks)

# 4 Task 4: Simple Average of the Mark

When priority equals 2, the chip will calculate the simple average of all MARK values stored in the RAM chips and output the value in **avg[16]**. This is simply done by dividing the sum of all MARKs by 4 (total register number). Turn on **Probe** and set **priority** = 2 to get the simple average of exam marks. For example, as shown in Fig.7, with the same data loaded in `Task 2`, the simple average mark **avg** is calculated as (79+97)/4 = 44.

| Input pins | | Output pins | |
|---|---|---|---|
| Name | Value | Name | Value |
| inID[16] | 0 | outID[16] | 0 |
| inMARK[16] | 0 | outMARK[16] | 0 |
| load | 0 | sum[16] | 0 |
| probe | 1 | avg[16] | 44 |
| address[2] | 0 | overflow | 0 |
| sl | 0 | | |
| priority[2] | 2 | | |

Figure 7: Example: Simple Average

**Input**:

1. A 2-bit switch for function priority called **priority**.
2. A 1-bit switch for probing the data called **probe**.

**Output**:

1. A 16-bit version of the average of the marks called **avg**.

(3 marks)

# 5 Task 5: Average of the Mark

When priority equals 3, the chip will calculate the real average of all MARK values stored in the RAM chips and output the value in **avg[16]**. To calculate the true average score, you need to consider the number of individuals in the database and exclude those with a student ID of 0 from the calculation (i.e., empty register). The average score should also be rounded down to the nearest integer. Turn on **Probe** and set **priority** = 3 to get the average of exam marks. For example, as shown in Fig.8, with the same data loaded in `Task 2`, the real average mark **avg** is calculated as (79+97)/2 = 88.

| Input pins | | Output pins | |
|---|---|---|---|
| Name | Value | Name | Value |
| inID[16] | 0 | outID[16] | 0 |
| inMARK[16] | 0 | outMARK[16] | 0 |
| load | 0 | sum[16] | 0 |
| probe | 1 | avg[16] | 88 |
| address[2] | 0 | overflow | 0 |
| sl | 0 | | |
| priority[2] | 3 | | |

Figure 8: Example: Average

**Input**:

1. A 2-bit switch for function priority called **priority**.
2. A 1-bit switch for probing the data called **probe**.

**Output**:

1. A 16-bit version of the average of the marks called **avg**.

(2 marks)

# Submission

You should zip all your files into one zip file to "Coursework1 Assignment". You should name your file as: `YOURSTUDENTID_YOURNAME`.zip. Your zip file should include:

1. one master hdl file called CW.hdl file
2. all additional hdl files used by CW.hdl
3. one readme.txt file if needed (optional)

Submit your zip file onto the Moodle submission page. Please note that every next submission overwrites all the files in the previous one. If you submit several times, make sure that your last submission includes all the necessary files. Include all required chips, instructions for use, and any instruction text file if necessary. For late submission, the standard late submission policy applies, i.e. 5% mark deduction for every 24 hours.