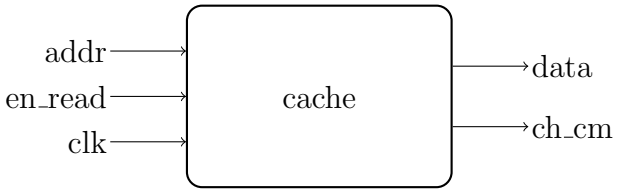
Cache

Ihre Aufgabe ist es, das Verhalten einer Entity namens „cache“ zu programmieren. Die Entity ist in der angehängten Datei „cache.vhdl“ deklariert und hat folgende Eigenschaften:

- Eingang: addr vom Typ std_logic_vector mit Länge 9
- Eingang: en_read vom Typ std_logic
- Eingang: clk vom Typ std_logic
- Ausgang: data vom Typ std_logic_vector mit Länge 5
- Ausgang: ch_cm vom Typ std_logic



Verändern Sie die Datei „cache.vhdl“ nicht!

Ziel dieser Aufgabe ist es, die Leseeinheit eines *direct mapped* Caches zu programmieren. Abhängig von der angelegten Adresse am Adresseingang *addr* des Caches soll dieser durchsucht und bei einem *cache hit* der Speicherinhalt ausgegeben werden. Bei einem *direct mapped* Cache ist die Position eines Datums im Cache eindeutig über die Adresse definiert. Umfasst der Cache M Einträge, so werden N = $\lceil log_2(M) \rceil$ Bits der Adresse für die Positionbestimmung (=Index) im Cache benötigt und die restlichen Bits der Adresse als *Tag* zu dem Datum im Cache dazugespeichert. Bei der Überprüfung, ob ein Datum im Cache vorhanden ist, muss also zunächst die Position im Cache anhand eines Teilstücks der Adresse ermittelt und anschließend der gespeicherte Tag mit dem zweiten Adressteil verglichen werden. Wenn dieser übereinstimmt, liegt ein *cache hit* vor. Abbildung 1 verdeutlicht das Prinzip eines *direct mapped* Caches.

Die Cache Entity soll dabei folgende Aufgaben erfüllen:

- Der Cache soll bei steigender Flanke des Takteinganges *clk* arbeiten, d.h. die Ausgänge dürfen sich nur bei einer steigenden Flanke verändern.
- Ein Zugriff auf den Cache erfolgt nur, wenn *en_read* gleich ’1’ ist, anderenfalls soll der Ausgang *data* immer hochomig (High Impedance = ’Z’) sein und das Flag *ch_cm* immer ’0’ liefern.
- Die im Cache gespeicherten 17 Daten und deren Tags sind in Tabelle 1 vorgegeben und sollen von Ihnen als konstanter Inhalt in Ihren Cache programmiert werden. (Diese würden von einer Schreibe-Einheit entsprechend beschrieben werden).

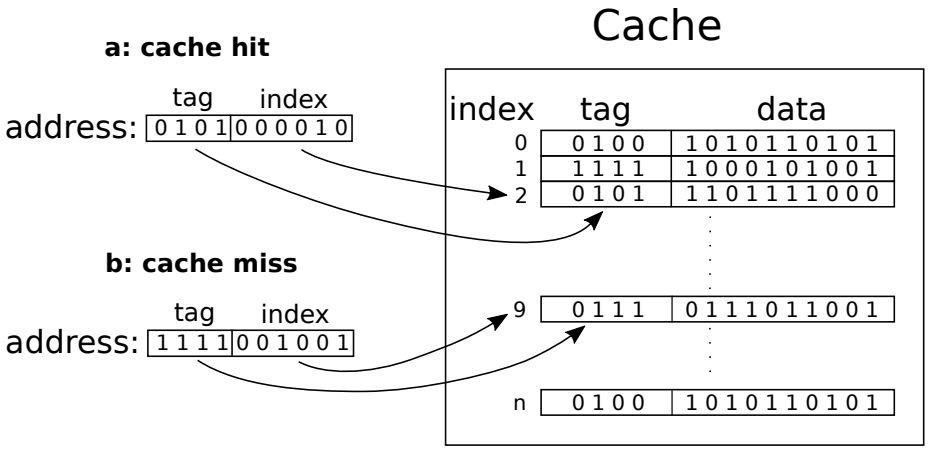


Abbildung 1: Prinzipdarstellung des Lesezugriffs bei einem *direct mapped* Cache. Dargestellt sind jeweils ein Beispiel für einen *cache hit* (a) und einen *cache miss* (b).
(Achtung: Die in dieser Darstellung verwendeten Bitbreiten stimmen nicht notwendigerweise mit jener aus Ihrer Angabe überein.)

- Erfolgt ein Lesezugriff (*en_read* = '1'), so soll aus der Adresse *addr* die mögliche Position des Datums im Cache (=Index) und der zugehörige Tag ermittelt werden. Kommt es dabei zu einem *cache hit*, so soll das Datum dieser Adresse am Ausgang *data* ausgegeben werden und das Flag *ch_cm* auf '1' gesetzt werden. Kommt es zu einem *cache miss*, so soll der Ausgang *data* hochomig (High Impedance = 'Z') sein und das Flag *ch_cm* '0' liefern.

- Ist bei einem Lesezugriff der Cache-Index der Adresse größer als die Speichertiefe des Caches, so soll ebenfalls der Ausgang *data* hochomig (High Impedance = 'Z') sein und das Flag *ch_cm* '0' liefern.

- Für die Berechnung des Cache-Index und des Tags aus der angelegten Adresse gilt außerdem: Der Index berechnet sich aus den n niederwertigsten Adressbits, wobei n die minimal nötige Anzahl an Bits zur Darstellung aller 17 Indizes im Cache ist. Die restlichen m Bits werden als Tag gespeichert (n + m = Adresslänge = 9).

Dieses Verhalten muss in der angehängten Datei „cache_beh.vhdl“ programmiert werden. Um Ihre Lösung abzugeben, senden Sie ein E-Mail mit dem Betreff „Result Task 6“ an vhdl-mc+e384@tuwien.ac.at und hängen Ihre Datei “cache_beh.vhdl” an.

Viel Erfolg und möge die Macht mit Ihnen sein.

Cache Index	Tag	Data
0	1010	11100
1	1111	11010
2	1101	10000
3	1010	10011
4	1000	11111
5	1101	11101
6	1011	10011
7	1011	11101
8	1000	10111
9	1011	10101
10	1110	11001
11	1001	10110
12	1001	11110
13	1000	11100
14	1101	10001
15	1001	11101
16	1101	10111

Tabelle 1: Inhalt des Caches. Die Daten und Tags sind jeweils mit MSB first angegeben.