Digital Logic

Design Assignment

2 (Course Project)

Submitted by

Name	:	Ekta Sai Meda
Register Number	:	220200019
E-mail ID	:	Meda Ekta Sai
School of Study	:	SCDS
Year of Study	:	2nd year

Q1:

1. Implementation

implements several functions to minimise Boolean expressions using the Quine-McCluskey algorithm. Here's an overview of the functions and their roles:

1. `generate_mints(input_str)`: Takes a string input and produces a list of minterms from the ASCII values of the uppercase characters.

2. `binary_rep(mint, num_vars)`: Converts a minterm into its binary representation using the specified number of variables.

3. `combine_terms(term1, term2)`: Combines two binary terms if they differ by only one bit, returning the combined term with a dash (-) in the differing bit's position, or `None` if they can't be combined.

4. `find_prime_imps(mints, num_vars)`: Identifies the prime implicants for a set of minterms. It converts minterms to binary, then iteratively combines pairs of terms differing by one bit. Terms left unmarked after iterations are prime implicants.

5. `find_ess_prime_imps(mints, prime_imps)`: Determines the essential prime implicants from a set of prime implicants and minterms. An implicant is essential if it uniquely covers a specific minterm.

6. `minimize_exp(mints, ess_prime_imps, num_vars, vars)`: Minimises the sum-of-products (SOP) expression using essential prime implicants, combining variable names and their complements based on the essential implicants.

The code then demonstrates usage through two parts:

Part (a):

- Uses the name "EKTA" to generate minterms.

- Computes the prime and essential prime implicants.

- Prints the minimised SOP expression.

Part (b):

- Provides a predefined list of minterms.

- Calculates the prime and essential prime implicants.

- Prints the minimised SOP expression.

- Adds an additional prime implicant for minterm 7 and prints the updated minimised expression.

To implement this, use Python 3 and the `itertools` module. Copy the code into a Python file (e.g., `quine_mccluskey.py`) and run it to see the minterms, prime implicants, essential prime implicants, and minimised SOP expressions for both parts (a) and (b).

The algorithm proceeds through the following steps:

1. Create a list of minterms from the input string by converting each uppercase character to its ASCII value and breaking down the individual digits.

2. Transform each minterm into its binary representation, considering the specified number of variables.

3. Identify prime implicants by iteratively combining pairs of binary terms that differ by only one bit.

4. Determine the essential prime implicants by identifying which prime implicants exclusively cover specific minterms.

5. Simplify the sum-of-products (SOP) expression by forming terms from the essential prime implicants, using the variable names and their complements.

6. Display the minterms, prime implicants, essential prime implicants, and the simplified SOP expression.

CODE:

```python
from itertools import combinations

# Define functions for handling minterms and prime implicants

def generate_mints(input_str):
    """
    Generate a list of minterms from a given input string.

    Args:
        input_str (str): The input string.

    Returns:
        list: A list of minterms.
    """
    ascii_codes = [ord(char) for char in input_str.upper()]  # Get ASCII codes of uppercase characters in the input string
    mints = set()
    for code in ascii_codes:
        for digit in str(code):
            mints.add(int(digit))  # Add individual digits of ASCII codes to the set of minterms
    return list(mints)
```

```python
def binary_rep(mint, num_vars):
    """
    Convert a minterm to its binary representation.

    Args:
        mint (int): The minterm value.
        num_vars (int): The number of variables.

    Returns:
        str: The binary representation of the minterm.
    """
    return format(mint, f'0{num_vars}b')  # Format the minterm as a binary string with leading zeros

def combine_terms(term1, term2):
    """
    Combine two binary terms if they differ by one bit.

    Args:
        term1 (str): The first binary term.
        term2 (str): The second binary term.

    Returns:
        str or None: The combined term if they differ by one bit, otherwise None.
    """
    diff = sum(1 for a, b in zip(term1, term2) if a != b)  # Count the number of differing bits
    if diff == 1:
        return ''.join('-' if a != b else a for a, b in zip(term1, term2))  # Combine the terms with '-' for differing bit
    return None
```

```python
def find_prime_imps(mints, num_vars):
    """
    Find the prime implicants for a given set of minterms.

    Args:
        mints (list): A list of minterms.
        num_vars (int): The number of variables.

    Returns:
        list: A list of prime implicants.
    """
    terms = [binary_rep(mint, num_vars) for mint in mints]  # Convert minterms to binary representations
    prime_imps = set()
    while terms:
        new_terms = set()
        marked = set()
        for term1, term2 in combinations(terms, 2):  # Try combining all pairs of terms
            combined = combine_terms(term1, term2)
            if combined:
                new_terms.add(combined)
                marked.add(term1)
                marked.add(term2)
        prime_imps.update(term for term in terms if term not in marked)  # Add unmarked terms to prime implicants
        terms = list(new_terms)  # Update terms with new combined terms
    return list(prime_imps)
```

```python
def minimize_exp(mints, ess_prime_imps, num_vars, vars):
    """
    Minimize the sum-of-products (SOP) expression using the essential prime implicants.

    Args:
        mints (list): A list of minterms.
        ess_prime_imps (list): A list of essential prime implicants.
        num_vars (int): The number of variables.
        vars (str): A string containing the variable names.

    Returns:
        str: The minimized SOP expression.
    """
    minimized_terms = []
    for implicant in ess_prime_imps:
        term = ''.join(
            vars[i] + ("'" if bit == '0' else '')
            for i, bit in enumerate(implicant) if bit != '-'
        )
        minimized_terms.append(term)
    minimized_exp = " + ".join(minimized_terms)
    return minimized_exp
```

```python
# Part (a)
name_a = "EKTA"
mints_a = generate_mints(name_a)
num_vars_a = 4
vars_a = "ABCD"
prime_imps_a = find_prime_imps(mints_a, num_vars_a)
ess_prime_imps_a = find_ess_prime_imps(mints_a, prime_imps_a)
minimized_exp_a = minimize_exp(mints_a, ess_prime_imps_a, num_vars_a, vars_a)

print("Part (a):")
print("Minterms:", mints_a)
print("Prime Implicants:", prime_imps_a)
print("Essential Prime Implicants:", ess_prime_imps_a)
print("Minimized SOP Expression:", minimized_exp_a)
print()
```

```python
# Part (b)
mints_b = [0, 5, 7, 8, 9, 12, 13, 23, 24, 25, 28, 29, 37, 40, 42, 44, 46, 55, 56, 57, 60, 61]
num_vars_b = 6
vars_b = "ABCDEF"
prime_imps_b = find_prime_imps(mints_b, num_vars_b)
ess_prime_imps_b = find_ess_prime_imps(mints_b, prime_imps_b)
minimized_exp_b = minimize_exp(mints_b, ess_prime_imps_b, num_vars_b, vars_b)

print("Part (b):")
print("Minterms:", mints_b)
print("Prime Implicants:")
for imp in prime_imps_b:
    print(imp)
print("Essential Prime Implicants:")
for imp in ess_prime_imps_b:
    print(imp)
print("Minimized SOP Expression:", minimized_exp_b)

# Adding the 7th prime implicant to the minimized expression for part (b)
additional_prime_imp = find_prime_imps([7], num_vars_b)
ess_prime_imps_b += additional_prime_imp
minimized_exp_b_updated = minimize_exp(mints_b, ess_prime_imps_b, num_vars_b, vars_b)

print("\nPart (b) with 7th Prime Implicant Added:")
print("Updated Minimized SOP Expression:", minimized_exp_b_updated)
```

Results:

```
Part (a):
Minterms: [4, 5, 6, 7, 8, 9]
Prime Implicants: ['100-', '01--']
Essential Prime Implicants: ['01--', '100-']
Minimized SOP Expression: A'B + AB'C'

Part (b):
Minterms: [0, 5, 7, 8, 9, 12, 13, 23, 24, 25, 28, 29, 37, 40, 42, 44, 46, 55, 56, 57, 60, 61]
Prime Implicants:
101--0
00-000
0001-1
-11-0-
--1-00
-10111
00-101
0-1-0-
0-0111
-00101
Essential Prime Implicants:
101--0
00-000
-11-0-
-10111
0-1-0-
-00101
Minimized SOP Expression: AB'CF' + A'B'D'E'F' + BCE' + BC'DEF + A'CE' + B'C'DE'F

Part (b) with 7th Prime Implicant Added:
Updated Minimized SOP Expression: AB'CF' + A'B'D'E'F' + BCE' + BC'DEF + A'CE' + B'C'DE'F + A'B'C'DEF
```

References:

-T. Agarwal, “Quine Mccluskey Method : Algorithm, Example & Its Applications,” ElProCus - Electronic Projects for Engineering Students, Jul. 25, 2023. https://www.elprocus.com/quine-mccluskey-method/ (accessed May 19, 2024).
-GeeksforGeeks, “Quine McCluskey Method,” GeeksforGeeks, Apr. 22, 2022. Accessed: May 19, 2024. [Online]. Available: https://www.geeksforgeeks.org/quine-mccluskey-method/ -M. Morris Mano, Digital Design. Pearson Educación, 2002.