EE 709 : BDD Assignment

Rohan Rajesh Kalbag
20D170033

April 2023

1 Describing the 4 bit adder

- The variables for the inputs are created as `bdd` arrays `x[4]`, `y[4]` and for the adder outputs `sum[4]` and `cout`, created using `bdd_new_var_last()` function.
- Temporary `bdd` arrays `s[4]` and `c[4]` are used to represent the actual sum and carry in terms of the input bits. They are recursively defined as follows following **Ripple Carry Adder Architecture**

$$s[0] = x[0] \oplus y[0] \tag{1}$$

$$c[0] = x[0] \cdot y[0] \tag{2}$$

$$s[i] = (x[i] \oplus y[i]) \oplus c[i-1], 1 \leq i \leq 3 \tag{3}$$

$$c[i] = x[i] \cdot y[i] + c[i-1] \cdot (x[i] \oplus y[i]), 1 \leq i \leq 3 \tag{4}$$

2 Question 1

2.1 Method

- Let A be the subset of the domain and $P(x, y)$ denote the function which generates this subset, i.e, $P(x, y) = 1 \iff$ there are odd number of bits 1 in both $x[:]$ and $y[:]$. Then $P(x, y)$ is represented as

$$P(x, y) = (x_0 \oplus x_1 \oplus x_2 \oplus x_3) \cdot (y_0 \oplus y_1 \oplus y_2 \oplus y_3) \tag{5}$$

- Let the equations (1), (2), (3), (4) be captured by $F(x, y)$
- The BDD of the image be represented as $Q(s, c_{out})$, which can be obtained using the existential quantification operation.

$$Q(s, c_{out}) = \exists_{x,y} P(x, y) \cdot (c_{out}, s \iff F(x, y)) \tag{6}$$

- Where $c_{out}, s \iff F(x, y)$ denotes equivalence

$$(c_{out} \iff c[3]) \cdot (sum[i] \iff s[i]), 0 \leq i \leq 3 \tag{7}$$

The entire code can be found in **Appendix** (5)

2.2 Image BDD Obtained

```
bdd of image, where var.8 - cout
var.9 - sum[0], var.10 - sum[1], var.11 - sum[2], var.12 - sum[3]

if var.8
  if var.9
    if var.10
      !var.11
    else if !var.10
      0: if var.11
        !var.12
      else if !var.11
        1
      endif var.11
    endif var.10
  else if !var.9
    if var.10
      subformula 0
    else if !var.10
      1
    endif var.10
  endif var.9
else if !var.8
  if var.9
    if var.10
      if var.11
        var.12
      else if !var.11
        1
      endif var.11
    else if !var.10
      1: if var.11
        1
      else if !var.11
        var.12
      endif var.11
    endif var.10
  else if !var.9
    if var.10
      1
    else if !var.10
      subformula 1
    endif var.10
  endif var.9
endif var.8
```

3 Question 2

3.1 Method

- Let B the subset of the range and let $B(s, c_{out})$ be the function which generates this function, i.e, $B(s, c) = 1 \iff$ there are odd number of bits 1 in output 5 bits. B is represented as

$$B(s, c_{out}) = c_{out} \oplus s_3 \oplus s_2 \oplus s_1 \oplus s_0 \tag{8}$$

- Let the BDD of the pre-image be represented as $S(x, y)$ which can be obtained by the existential quantification operation

$$S(x, y) = \exists_{s,c} B(s, c) \cdot (s, c \iff F(x, y)) \tag{9}$$

- Where $c_{out}, s \iff F(x, y)$ denotes equivalence

$$(c_{out} \iff c[3]) \cdot (sum[i] \iff s[i]), 0 \leq i \leq 3 \tag{10}$$

The code for the implementation can be found in **Appendix** (5)

3.2 Pre-Image BDD Obtained

```
bdd of pre-image, where var.0 - x[0]
var.1 - x[1], var.2 - x[2], var.3 - x[3], var.4 - y[0]
var.5 - y[1], var.6 - y[2], var.7 - y[3]
----------------------------------------------------
if var.0
  if var.1
    if var.2
      if var.3
        if var.4
          0: if var.5
            1: if var.6
              !var.7
            else if !var.6
              var.7
            endif var.6
          else if !var.5
            !subformula 1
          endif var.5
        else if !var.4
          2: if var.5
            subformula 1
          else if !var.5
            3: if var.6
              !var.7
            else if !var.6
              0
            endif var.6
```

```
      endif var.5
    endif var.4
  else if !var.3
    if var.4
      4: if var.5
        var.6
      else if !var.5
        !var.6
      endif var.5
    else if !var.4
      5: if var.5
        var.6
      else if !var.5
        6: if var.6
          1
        else if !var.6
          !var.7
        endif var.6
      endif var.5
    endif var.4
  endif var.3
else if !var.2
  if var.3
    if var.4
      7: if var.5
        !subformula 3
      else if !var.5
        subformula 3
      endif var.5
    else if !var.4
      8: if var.5
        !subformula 3
      else if !var.5
        !var.6
      endif var.5
    endif var.4
  else if !var.3
    if var.4
      9: if var.5
        !subformula 6
      else if !var.5
        subformula 6
      endif var.5
    else if !var.4
      10: if var.5
        !subformula 6
      else if !var.5
        subformula 1
```

```
          endif var.5
        endif var.4
      endif var.3
    endif var.2
  else if !var.1
    if var.2
      if var.3
        if var.4
          !subformula 2
        else if !var.4
          !subformula 7
        endif var.4
      else if !var.3
        if var.4
          !subformula 5
        else if !var.4
          !subformula 9
        endif var.4
      endif var.3
    else if !var.2
      if var.3
        if var.4
          !subformula 8
        else if !var.4
          !subformula 4
        endif var.4
      else if !var.3
        if var.4
          !subformula 10
        else if !var.4
          subformula 0
        endif var.4
      endif var.3
    endif var.2
  endif var.1
else if !var.0
  if var.1
    if var.2
      if var.3
        if var.4
          subformula 2
        else if !var.4
          !subformula 2
        endif var.4
      else if !var.3
        if var.4
          subformula 5
        else if !var.4
```

```
      !subformula 5
    endif var.4
  endif var.3
else if !var.2
  if var.3
    if var.4
      subformula 8
    else if !var.4
      !subformula 8
    endif var.4
  else if !var.3
    if var.4
      subformula 10
    else if !var.4
      !subformula 10
    endif var.4
  endif var.3
endif var.2
else if !var.1
if var.2
  if var.3
    if var.4
      !subformula 7
    else if !var.4
      subformula 7
    endif var.4
  else if !var.3
    if var.4
      !subformula 9
    else if !var.4
      subformula 9
    endif var.4
  endif var.3
else if !var.2
  if var.3
    if var.4
      !subformula 4
    else if !var.4
      subformula 4
    endif var.4
  else if !var.3
    if var.4
      subformula 0
    else if !var.4
      !subformula 0
    endif var.4
  endif var.3
endif var.2
```

```
  endif var.1
endif var.0
```

4 Question 3

4.1 Method

Note: The claim *every even 4-bit number (n) can be expressed as a sum of two prime numbers*, is true only if the number is greater than 2. **Assumption:** $n > 2$

- Let $s : (s_3, s_2, s_1, s_0)$ denote the four bit number, let it be the sum of two four bit numbers $x : (x_3, x_2, x_1, x_0)$ and $y : (y_3, y_2, y_1, y_0)$
- Let R the subset of the range such that $s > 2$ is even and let $R(s)$ be the function which generates this function. $s \in \{4, 6, 8, 10, 12, 14\}$. On solving K-Map we get

$$R(s) = \bar{s_0} \cdot (s_2 + s_3) \tag{11}$$

- Let M be the subset of the domain such that x, y are both prime. Let $M(x, y)$ be the generating function for this. $x, y \in \{2, 3, 5, 7, 11, 13\}$. On solving K-Map we get $M(x, y)$ as

$$(x_0 \cdot x_1 \cdot \bar{x_2} + \bar{x_3} \cdot \bar{x_2} \cdot x_1 + \bar{x_3} \cdot x_2 \cdot x_0 + \bar{x_1} \cdot x_0 \cdot x_2) \cdot (y_0 \cdot y_1 \cdot \bar{y_2} + \bar{y_3} \cdot \bar{y_2} \cdot y_1 + \bar{y_3} \cdot y_2 \cdot y_0 + \bar{y_1} \cdot y_0 \cdot y_2) \tag{12}$$

- The claim can be expressed as the following logical expression

$$\forall_s (R(s) \to \exists_{x,y} M(x, y) \cdot (s \iff F(x, y))) \tag{13}$$

- To prove the claim, we need to show that the above expression is equivalent to a tautology.
- Where $s \iff F(x, y)$ denotes equivalence

$$(sum[i] \iff s[i]), 0 \leq i \leq 3 \tag{14}$$

4.2 Implementation

This `if-else` block checks if the final BDD for the logical expression equals to the bdd for one.

```
if(final == bdd_one(bddm)){
    printf("The claim is correct, Hence Proved!\n");
}
else{
    printf("The claim is incorrect, Hence Disproved!\n");
}
```

The entire code can be found in **Appendix** (5)

4.3 Output Obtained

```
The claim is correct, Hence Proved!
```

5 Appendix for Code

5.1 fourbitadder.c

```
#include <stdlib.h>
#include <stdio.h>
#include <bdduser.h>

// number of bits in adder


void visualize_bdd(bdd_manager bddm, bdd x){
        printf("---------------------------------------------------\n");
        bdd_print_bdd(bddm,x,NULL, NULL,NULL, stdout); //prints bdd
        printf("---------------------------------------------------\n");
}


int main (int argc, char* argv[])
{
        // create the universe

        bdd_manager bddm = bdd_init();

        // make inputs of adder

        bdd x[4];
        for(int i=0;i<4;i++){
                x[i] = bdd_new_var_last(bddm);
        }

        bdd y[4];
        for(int i=0;i<4;i++){
                y[i] = bdd_new_var_last(bddm);
        }

        // sum and carry of adder

        bdd s[4];
        bdd c[4];

        // first bit of adder

        s[0] = bdd_xor(bddm, x[0], y[0]);
        c[0] = bdd_and(bddm, x[0], y[0]);

        // subsequent rippling of adder bits

        for(int i=1; i<4; i++){
                s[i] = bdd_xor(bddm, x[i], y[i]);
                bdd temp = bdd_and(bddm, s[i], c[i-1]);
                s[i] = bdd_xor(bddm, s[i], c[i-1]);
```

```
        c[i] = bdd_and(bddm, x[i], y[i]);
        c[i] = bdd_or(bddm, c[i], temp);
}

bdd cout = bdd_new_var_last(bddm);
bdd sum[4];

for(int i=0; i<4; i++){
        sum[i] = bdd_new_var_last(bddm);
}

// approach to question 1

bdd x_has_odd_high_bits = bdd_xor(bddm, x[0], x[1]);

for(int i=2; i<4; i++){
        x_has_odd_high_bits = bdd_xor(bddm, x[i], x_has_odd_high_bits);
}

bdd y_has_odd_high_bits = bdd_xor(bddm, y[0], y[1]);

for(int i=2; i<4; i++){
        y_has_odd_high_bits = bdd_xor(bddm, y[i], y_has_odd_high_bits);
}

// function P(x, y) for subset A

bdd has_odd_high_bits = bdd_and(bddm, x_has_odd_high_bits, y_has_odd_high_bits);

// function equivalence check c, s <-> F(x, y)

bdd equivalence = bdd_xnor(bddm, cout, c[4 - 1]);

for(int i=0; i<4; i++){
        bdd curr_eq = bdd_xnor(bddm, s[i], sum[i]);
        equivalence = bdd_and(bddm, equivalence, curr_eq);
}

bdd condition = bdd_and(bddm, equivalence, has_odd_high_bits);

// To get image apply existence quantification
// Q(c, s) = Exists x,y P(x, y).(c, s <-> F(x))

bdd quantified_vars[9];

for(int i=0; i<4;i++){
        quantified_vars[i] = x[i];
}

for(int i=4; i<8; i++){
        quantified_vars[i] = y[i - 4];
}
```

```c
quantified_vars[8] = 0;

int assoc = bdd_new_assoc(bddm, quantified_vars, 0);
bdd_assoc(bddm, assoc);

// bdd for the image of the set

bdd image = bdd_exists(bddm, condition);

printf("bdd of image, where var.8 - cout\n");
printf("var.9 - sum[0], var.10 - sum[1], var.11 - sum[2], var.12 - sum[3]\n");
visualize_bdd(bddm, image);

bdd_free_assoc(bddm, assoc);

// approach for question 2

// bdd for B(s, c)

bdd output_odd_high_bits = bdd_xor(bddm, sum[0], cout);

for(int i=1; i<4; i++){
        output_odd_high_bits = bdd_xor(bddm, sum[i], output_odd_high_bits);
}

// to find the preimage of set B

condition = bdd_and(bddm, equivalence, output_odd_high_bits);

// To get pre-image apply existence quantification
// S(x, y) = Exists s,c B(s, c).(s, c <-> F(x, y))

for(int i=0; i<4;i++){
        quantified_vars[i] = sum[i];
}

quantified_vars[4] = cout;

quantified_vars[5] = 0;

assoc = bdd_new_assoc(bddm, quantified_vars, 0);
bdd_assoc(bddm, assoc);

// bdd for the pre-image of the set

bdd preimage = bdd_exists(bddm, condition);

printf("bdd of pre-image, where var.0 - x[0]\n");
printf("var.1 - x[1], var.2 - x[2], var.3 - x[3], var.4 - y[0]\n");
printf("var.5 - y[1], var.6 - y[2], var.7 - y[3]\n");
visualize_bdd(bddm, preimage);
```

```
	bdd_free_assoc(bddm, assoc);

	// approach for question 3

	bdd even_four_bit_num = bdd_and(bddm, bdd_not(bddm, sum[0]), bdd_or(bddm, sum[2],
↪   sum[3])); // R(s)

	bdd x_is_prime = bdd_or(bddm, bdd_and(bddm, x[0], bdd_xor(bddm, x[1], x[2])),
↪   bdd_and(bddm, bdd_not(bddm, x[3]), bdd_or(bddm, bdd_and(bddm, x[0], x[2]),
↪   bdd_and(bddm, x[1], bdd_not(bddm, x[2])))));
	bdd y_is_prime = bdd_or(bddm, bdd_and(bddm, y[0], bdd_xor(bddm, y[1], y[2])),
↪   bdd_and(bddm, bdd_not(bddm, y[3]), bdd_or(bddm, bdd_and(bddm, y[0], y[2]),
↪   bdd_and(bddm, y[1], bdd_not(bddm, y[2])))));
	bdd x_is_prime_and_y_is_prime = bdd_and(bddm, x_is_prime, y_is_prime); // M(x, y)

	// s <-> F(x, y)
	equivalence = bdd_xnor(bddm, s[0], sum[0]);

	for(int i=1; i<4; i++){
		equivalence = bdd_and(bddm, equivalence, bdd_xnor(bddm, s[i], sum[i]));
	}

	for(int i=0; i<4;i++){
		quantified_vars[i] = x[i];
	}

	for(int i=4; i<8; i++){
		quantified_vars[i] = y[i - 4];
	}

	quantified_vars[8] = 0;

	assoc = bdd_new_assoc(bddm, quantified_vars, 0);
	bdd_assoc(bddm, assoc);

	// M(x,y).(s <-> F(x, y))

	condition = bdd_and(bddm, equivalence, x_is_prime_and_y_is_prime);

	// Exists x,y M(x,y).(s <-> F(x, y))
	bdd exists = bdd_exists(bddm, condition);

	// R(s) -> Exists x,y M(x,y).(s <-> F(x, y))
	bdd implication = bdd_or(bddm, exists, bdd_not(bddm, even_four_bit_num));

	bdd_free_assoc(bddm, assoc);

	// to prove for all outputs
	for(int i=0;i<4;i++){
		quantified_vars[i] = sum[i];
	}
```

```c
        quantified_vars[4] = 0;

        assoc = bdd_new_assoc(bddm, quantified_vars, 0);
        bdd_assoc(bddm, assoc);

        // Forall s [ R(s) -> Exists x,y M(x,y).(s <-> F(x, y)) ]
        bdd final = bdd_forall(bddm, implication);

        if(final == bdd_one(bddm)){
                printf("The claim is correct, Hence Proved!\n");
        }
        else{
                printf("The claim is incorrect, Hence Disproved!\n");
        }

        bdd_free_assoc(bddm, assoc);
        bdd_quit(bddm);
        return(0);
}
```

5.2 compile.sh

```
gcc  -o fourbitadder fourbitadder.c -I ../cmu_bdd/include -L ../cmu_bdd/lib -lbdd -lmem
./fourbitadder > outputs.txt
```

6 Submission

The submission contains the .pdf report, .c source file, `fourbitadder` executable file and the outputs file `outputs.txt`

- In this case, the `fourbitadder.c` is kept in the following directory structure.

```
_
|_ cmu_bdd/
|_ folder/
    |_ compile.sh
    |_ fourbitadder.c
```

- run the bash script as `./compile.sh` to give final directory structure

```
_
|_ cmu_bdd/
|_ folder/
    |_ compile.sh
    |_ fourbitadder.c
    |_ fourbitadder
    |_ output.txt
```