Exercise 1

We need a software implementation of the traffic light controller to be used as the golden model.
The following code represents this model. Each *if*-statement represents one of the rules in the traffic light controller.

```
void traffic_light_sotfware(
               bool  l_sensor,
               bool  r_sensor,
               bool  u_sensor,
               bool  d_sensor,
               bool *ns_light,
               bool *ew_light )
{

  //rule 1
  if (r_sensor == 1 && l_sensor==1) //both lanes R and L are occupied
  {
    *ew_light = 1;
    *ns_light = 0;
  }
 //rule 2
  if (((r_sensor == 1 || l_sensor==0) || (r_sensor == 0 || l_sensor== 1))  //one of lanes R 
and L is occupied
      && (u_sensor == 0 && d_sensor == 0)  //both lanes U and D are not occupied
    )
  {
    *ew_light = 1;
    *ns_light = 0;
  }
  // rule 3
  if ((u_sensor == 1 && d_sensor == 1) //both lanes U and D are occupied
     && !(r_sensor == 1 && l_sensor==1) //cars are not detected on both lanes L and R
  )
  {
    *ew_light = 0;
    *ns_light = 1;
  }
  //rule 4
  if (((u_sensor == 1 || d_sensor == 0) || (u_sensor == 0 || d_sensor == 1)) //one of the 
lanes U and D is occupied
    && (r_sensor == 0 && l_sensor== 0) //  both lanes L and R are vacant
  )
  {
    *ew_light = 0;
    *ns_light = 1;
  }
  //rule 5
  if ((r_sensor == 0 || l_sensor==0) && (u_sensor == 0 && d_sensor == 0)) //all lanes are
vacant
  {
    *ew_light = 1;
    *ns_light = 0;
  }
}
```

Then we should develop the main function. The main function has three tasks

1- Generating test vectors (or test data)
2- Applying the test vectors to hardware description and software model
3- Compare the results and report any discrepancy

First, we define a status varible to keep the test result.

```
int main() {
	int status = 0;
	…
	…
	…
	if (status == 0) {
		std::cout << "Test Passed" << std::endl;
	} else {
		std::cout << "Test Failed" << std::endl;
	}
	return status;
}
```

Then we should generate test data. As the design input space is small we can generate all the possible inputs using four nested loops.

```
for (int l = 0; (l < 2) && (status == 0); l++) {
  for (int r = 0; (r < 2) && (status == 0); r++) {
    for (int u = 0; (u < 2) && (status == 0); u++) {
      for (int d = 0; (d < 2) && (status == 0); d++) {
        l_sensor = l;
        r_sensor = r;
        u_sensor = u;
        d_sensor = d;
        ...
        ...
        ...

      }
    }
  }
}
```

Applying each test vector to hardware and software implementation and comparing the results can be like this

```
traffic_light(l_sensor, r_sensor, u_sensor, d_sensor, &ns_light_hw, &ew_light_hw);
traffic_light_sotfware(l_sensor, r_sensor, u_sensor, d_sensor, &ns_light_sw, &ew_light_sw);

if ((ns_light_hw != ns_light_sw) || (ew_light_hw != ew_light_sw)) {

  status = -1;
  std::cout << "Error at "
           << " l_sensor = " <<     l_sensor << " r_sensor = " << r_sensor
            << " u_sensor = " << u_sensor
           << " d_sensor = " << d_sensor
           << " ns_light_hw = " << ns_light_hw << " ns_light_sw = " << ns_light_sw
           << " ew_light_hw = " << ew_light_hw << " ew_light_sw = " << ew_light_sw
           << std::endl;
}
```

Exercise 2

To solve this exercise, we use only the concepts that we have learned by now. However, there is a more straightforward approach to solve the problem in HLS, which we will learn along the course.

For the software implementation, we use integer addition. However, to be able to compare the result with the hardware, we should be able to split a 4-bit integer into separate bits. For this purpose, we use the split_bits function as follows.

```
void split_bits(int a, bool &a0, bool &a1, bool &a2, bool &a3) {

  a0 = a%2;
  a = a/2;

  a1 = a%2;
  a = a/2;

  a2 = a%2;
  a = a/2;

  a3 = a%2;
}
```

Now this is the software implementation

```
void adder_software(int a, int b, bool &s0, bool &s1, bool &s2, bool &s3, bool &c)
{
  int sum = a+b;
  if (sum < 16) { // extract the carry
    c=0;
  } else {
    sum = sum-16;
    c = 1;
  }
  split_bits(sum, s0, s1, s2, s3);
}
```

And this is the test bench main function

```
int main () {
  int status = 0;

  bool a0, a1, a2, a3;
  bool b0, b1, b2, b3;
  int a, b;
  bool s0_hw, s1_hw, s2_hw, s3_hw;
  bool s0_sw, s1_sw, s2_sw, s3_sw;

  bool c_hw, c_sw;

  for (int a = 0; a < 16 && status == 0; a++) {
    for (int b = 0; b < 16 && status == 0; b++) {
      split_bits(a, a0, a1, a2, a3);
      split_bits(b, b0, b1, b2, b3);

      fourbit_adder(a0, a1, a2, a3, b0, b1, b2, b3, &s0_hw, &s1_hw, &s2_hw, &s3_hw, &c_hw);
      adder_software(a, b, s0_sw, s1_sw, s2_sw, s3_sw, c_sw);

      if (s0_hw != s0_sw
          ||s1_hw != s1_sw
          ||s2_hw != s2_sw
          ||s3_hw != s3_sw
          ||c_hw != c_sw
      )
      {
        status = -1;
        std::cout << "Error at " << "a = " << a << " b = " << b
                << " a = " << a3 << a2 << a1 << a0
                << " b = " << b3 << b2 << b1 << b0
                << " s_hw = " << s3_hw << s2_hw << s1_hw << s0_hw
                << " s_sw = " << s3_sw << s2_sw << s1_sw << s0_sw
                << " c_hw = " << c_hw
                << " c_sw = " << c_sw
                << std::endl;
      }

    }
  }

  if (status == 0) {
    std::cout << "Test Passed" << std::endl;
  } else {
    std::cout << "Test Failed" << std::endl;
  }
  return status;
}
```

Exercise 3

This is the software implementation

```
void comparator_sw(int a, int b, bool &m_sw, bool &n_sw, bool &p_sw) {
  if (a == b) {
    m_sw = 1;
    n_sw = 0;
    p_sw = 0;
  }
  if (a > b) {
    m_sw = 0;
    n_sw = 1;
    p_sw = 0;
  }
  if (a < b) {
    m_sw = 0;
    n_sw = 0;
    p_sw = 1;
  }
}
```

And this is the test bench main function

```
int main() {
  int status = 0;

  int a;
  int b;

  bool a0, a1, a2, a3;
  bool b0, b1, b2, b3;

  bool m_hw, n_hw, p_hw;
  bool m_sw, n_sw, p_sw;

  for (int i = 0; (i < 16) && (status == 0) ; i++) {
    for (int j = 0; (j < 16) && (status == 0); j++) {
      a = i;
      b = j;
      split_bits(a, a0, a1, a2, a3);
      split_bits(b, b0, b1, b2, b3);
      comparator( a0, a1, a2, a3, b0, b1, b2, b3, m_hw, n_hw, p_hw);
      comparator_sw( a, b, m_sw, n_sw, p_sw);

      if ( (m_hw != m_sw) || (n_hw != n_sw) || (p_hw != p_sw) ){
        status = -1;
        std::cout << " Errot at "
                  << " a = " << a
                  << " b = " << b
                  << " m_hw = " << m_hw << "m_sw = " << m_sw
                  << " n_hw = " << n_hw << "n_sw = " << n_sw
                  << " p_hw = " << p_hw << "p_sw = " << p_sw
                  << std::endl;
      }

    }
  }
  if (status == 0 ) {
    std::cout << "OK " << std::endl;
  } else {
    std::cout << "Error " << std::endl;
  }

  return status;
}
```