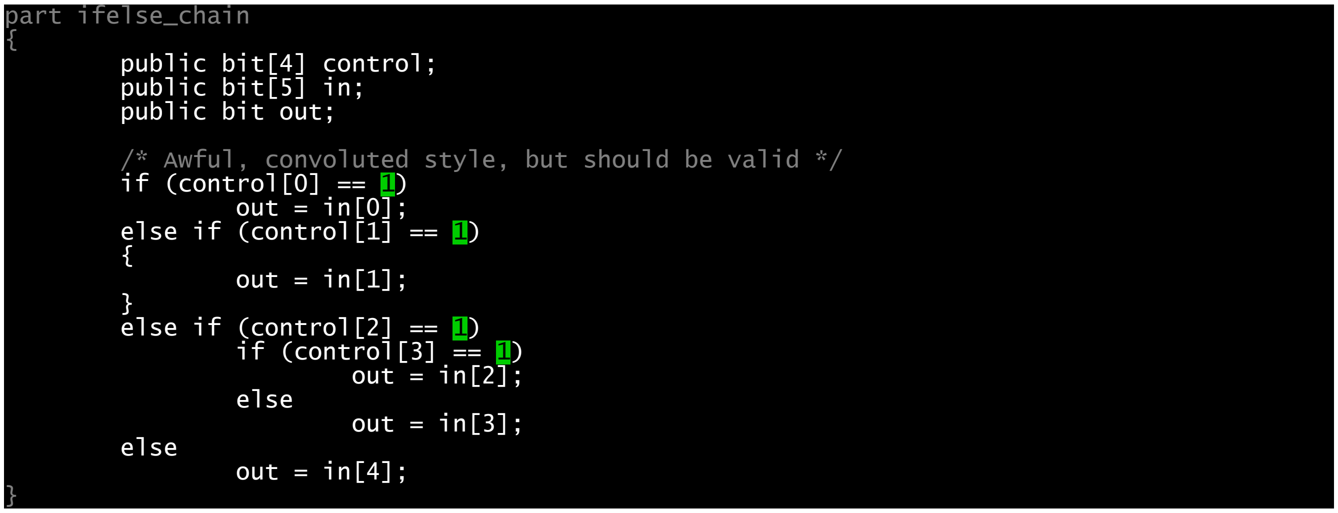
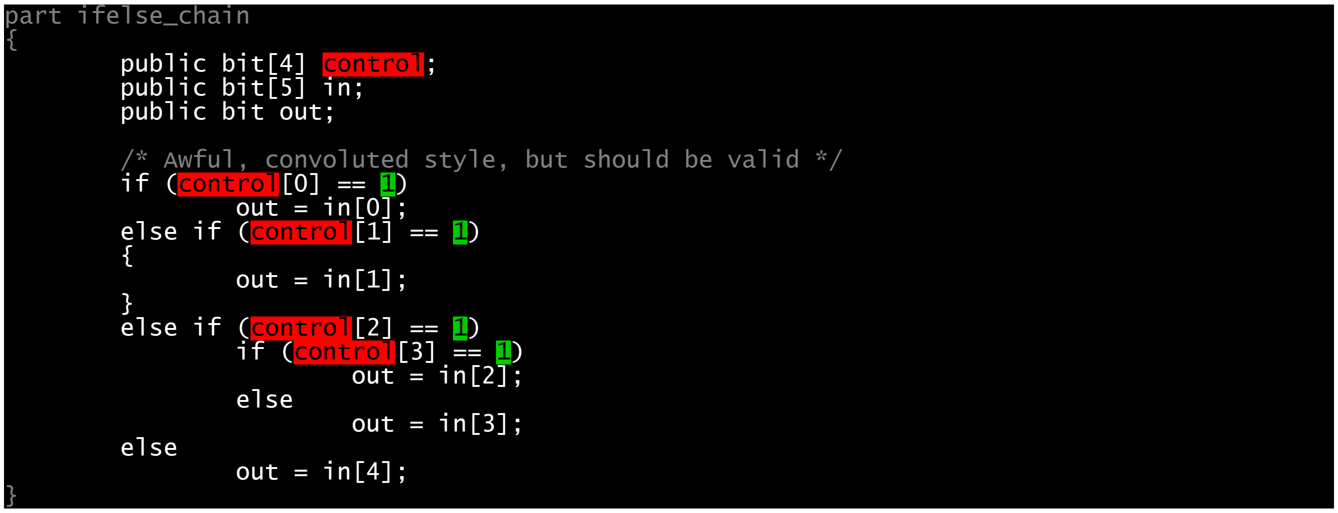


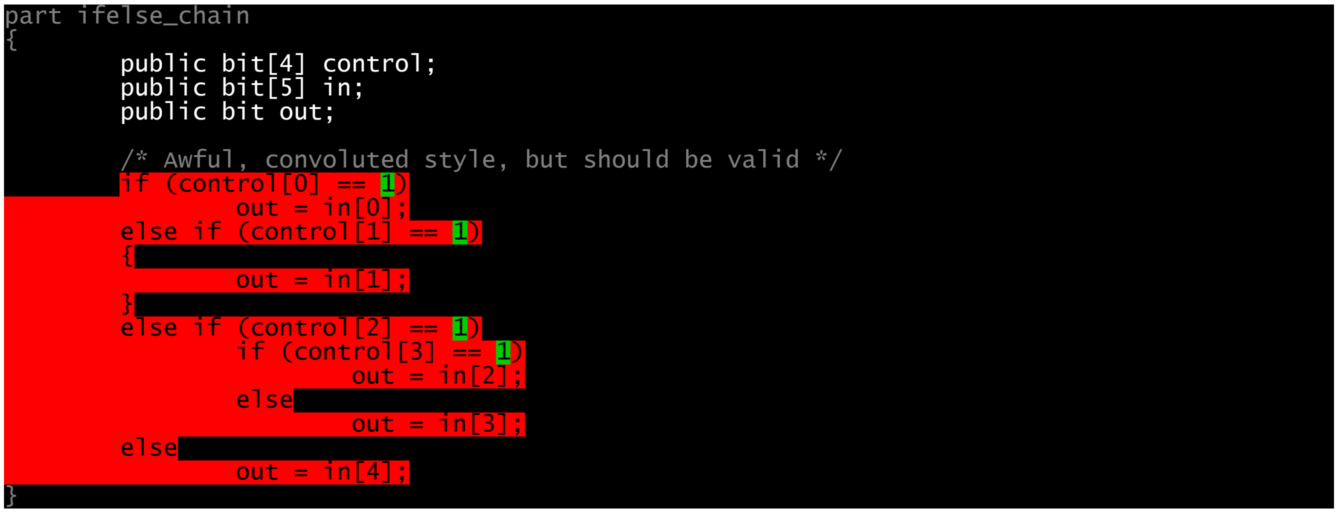
```
part ifelse_chain
{
        public bit[4] control;
        public bit[5] in;
        public bit out;

        /* Awful, convoluted style, but should be valid */
        if (control[0] == 1)
                out = in[0];
        else if (control[1] == 1)
        {
                out = in[1];
        }
        else if (control[2] == 1)
                if (control[3] == 1)
                        out = in[2];
                else
                        out = in[3];
        else
                out = in[4];
}
```

REMINDER: Let's use “jiffy” as the unit of time in which one step in the simulation occurs.



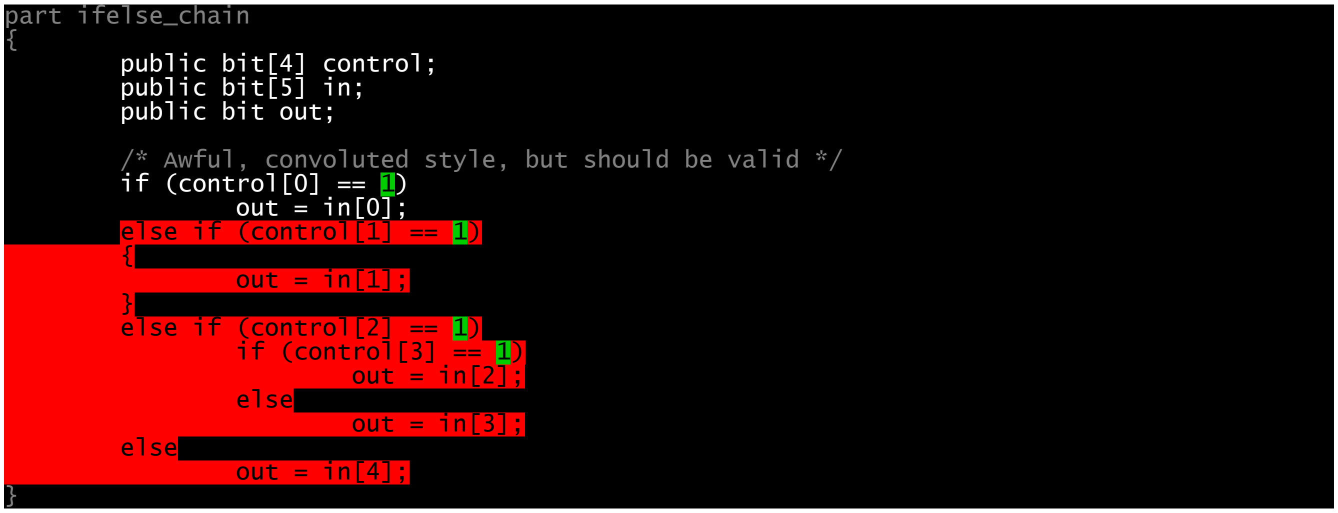
```
part ifelse_chain
{
        public bit[4] control;
        public bit[5] in;
        public bit out;

        /* Awful, convoluted style, but should be valid */
        if (control[0] == 1)
                out = in[0];
        else if (control[1] == 1)
        {
                out = in[1];
        }
        else if (control[2] == 1)
                if (control[3] == 1)
                        out = in[2];
                else
                        out = in[3];
        else
                out = in[4];
}
```

Mouse-over example:
The user mouses over 'control' declaration on line 3.



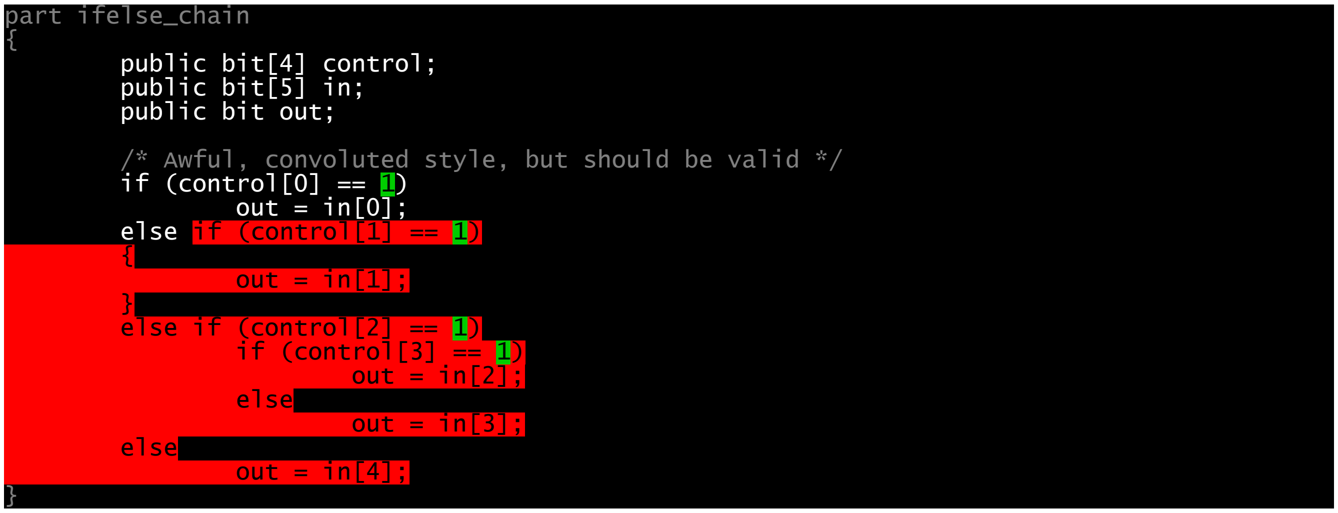
```
part ifelse_chain
{
        public bit[4] control;
        public bit[5] in;
        public bit out;

        /* Awful, convoluted style, but should be valid */
        if (control[0] == 1)
                out = in[0];
        else if (control[1] == 1)
        {
                out = in[1];
        }
        else if (control[2] == 1)
                if (control[3] == 1)
                        out = in[2];
                else
                        out = in[3];
        else
                out = in[4];
}
```

Mouse-over example:
The user mouses over the 'if' keyword on line 8.



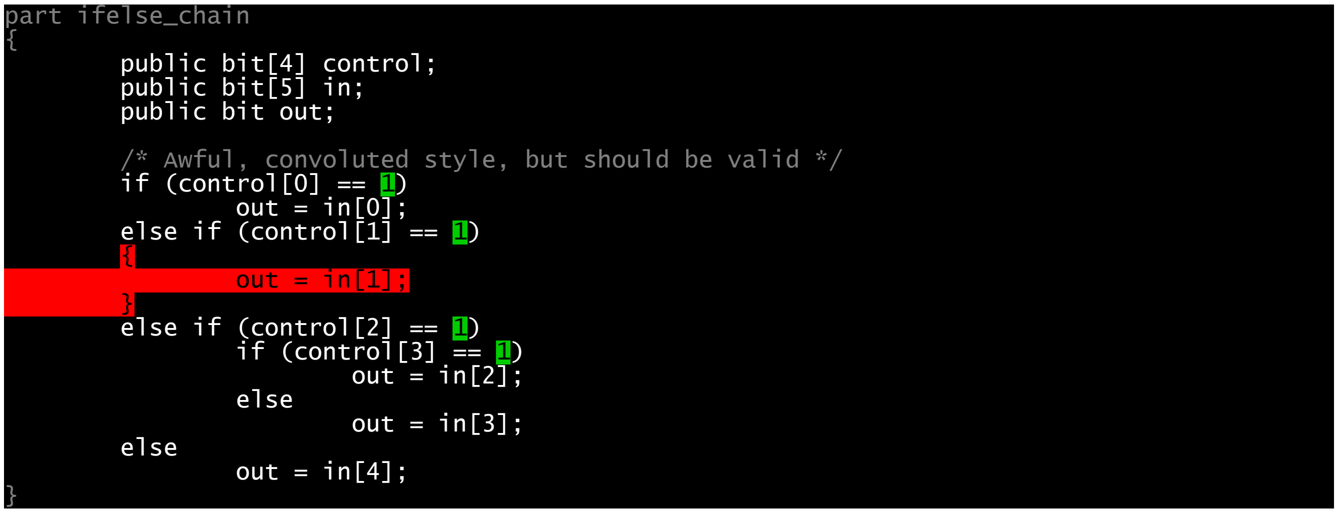
```
part ifelse_chain
{
        public bit[4] control;
        public bit[5] in;
        public bit out;

        /* Awful, convoluted style, but should be valid */
        if (control[0] == 1)
                out = in[0];
        else if (control[1] == 1)
        {
                out = in[1];
        }
        else if (control[2] == 1)
                if (control[3] == 1)
                        out = in[2];
                else
                        out = in[3];
        else
                out = in[4];
}
```

Mouse-over example:
The user mouses over the 'else' keyword on line 10.



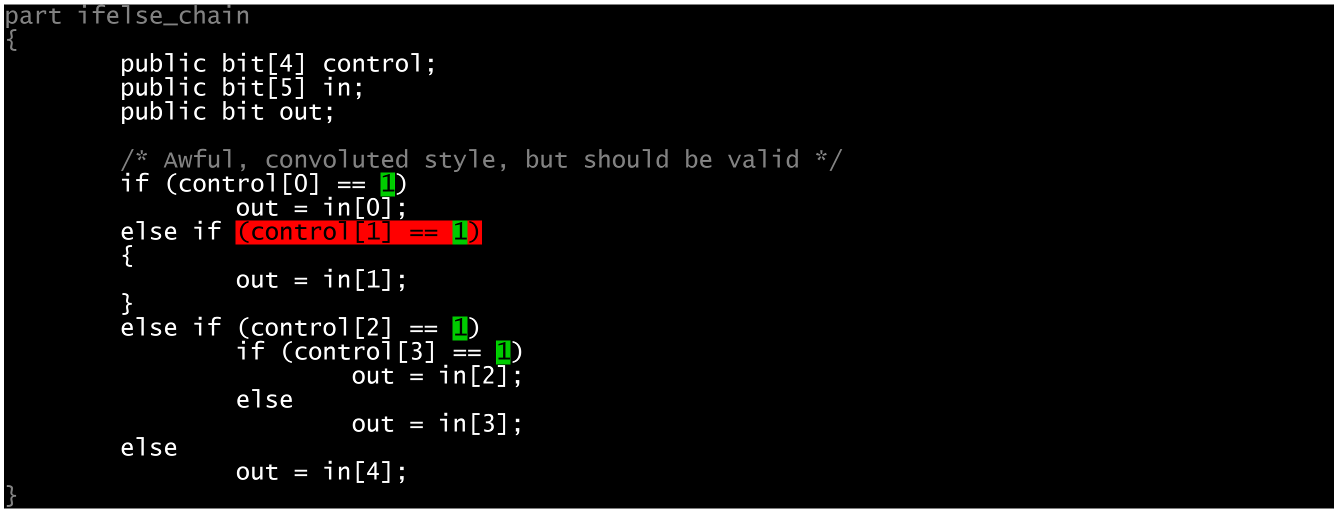
```
part ifelse_chain
{
        public bit[4] control;
        public bit[5] in;
        public bit out;

        /* Awful, convoluted style, but should be valid */
        if (control[0] == 1)
                out = in[0];
        else if (control[1] == 1)
        {
                out = in[1];
        }
        else if (control[2] == 1)
                if (control[3] == 1)
                        out = in[2];
                else
                        out = in[3];
        else
                out = in[4];
}
```

Mouse-over example:
The user mouses over the 'if' keyword on line 11.



```
part ifelse_chain
{
	public bit[4] control;
	public bit[5] in;
	public bit out;

	/* Awful, convoluted style, but should be valid */
	if (control[0] == 1)
		out = in[0];
	else if (control[1] == 1)
	{
		out = in[1];
	}
	else if (control[2] == 1)
		if (control[3] == 1)
			out = in[2];
		else
			out = in[3];
	else
		out = in[4];
}
```

Mouse-over example:
The user mouses over the { on line 12.

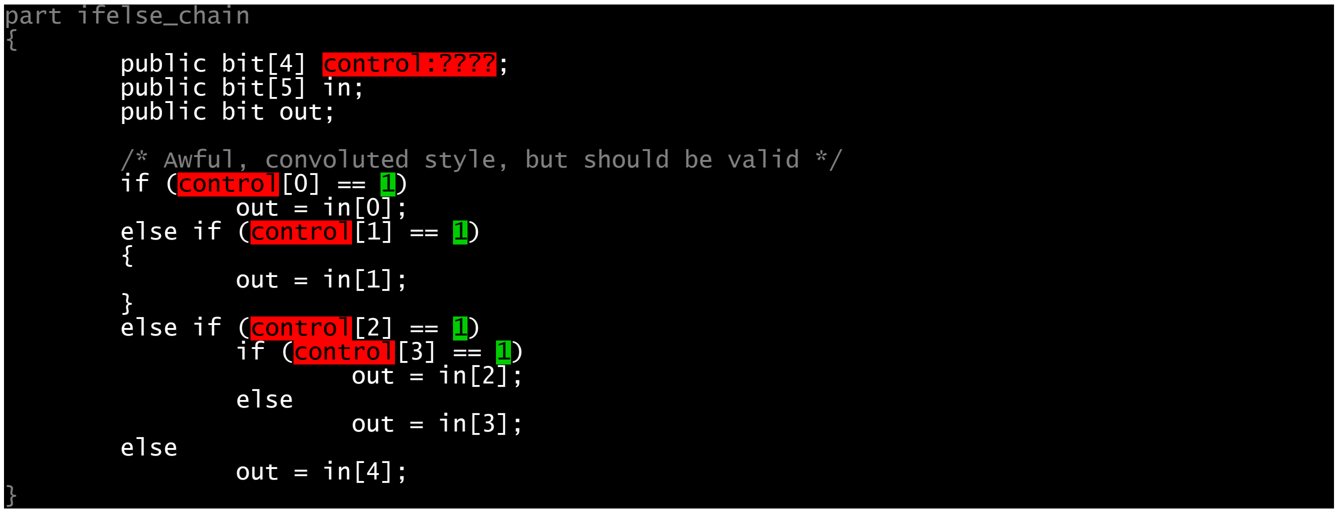


```
part ifelse_chain
{
        public bit[4] control;
        public bit[5] in;
        public bit out;

        /* Awful, convoluted style, but should be valid */
        if (control[0] == 1)
                out = in[0];
        else if (control[1] == 1)
        {
                out = in[1];
        }
        else if (control[2] == 1)
                if (control[3] == 1)
                        out = in[2];
                else
                        out = in[3];
        else
                out = in[4];
}
```

Mouse-over example:
The user mouses over the (on line 10.

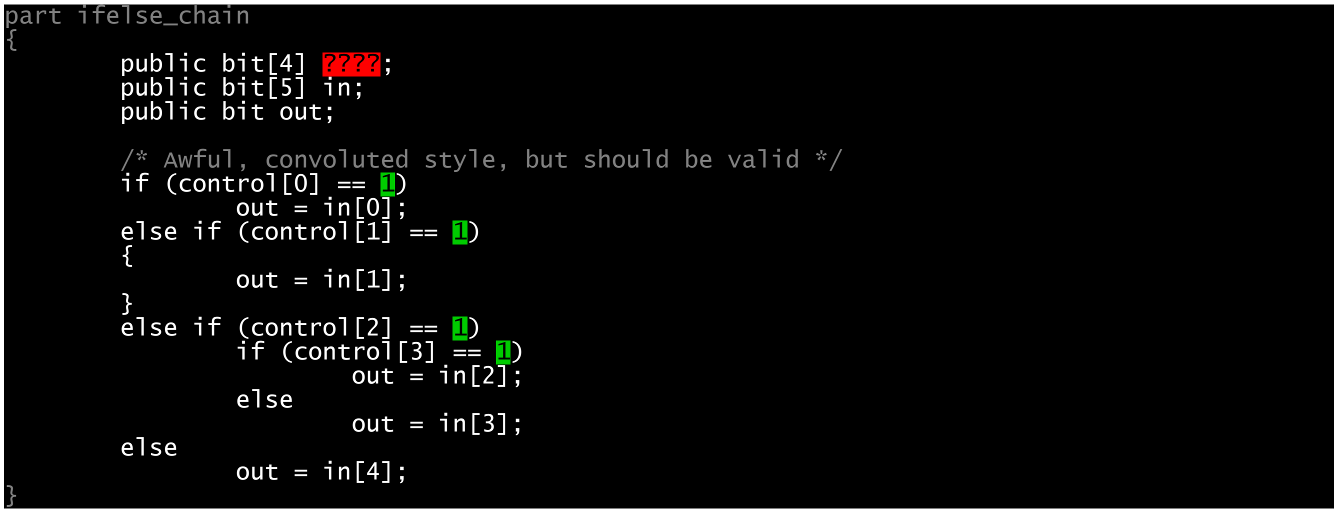
```
part ifelse_chain
{
	public bit[4] control;
	public bit[5] in;
	public bit out;

	/* Awful, convoluted style, but should be valid */
	if (control[0] == 1)
		out = in[0];
	else if (control[1] == 1)
	{
		out = in[1];
	}
	else if (control[2] == 1)
		if (control[3] == 1)
			out = in[2];
		else
			out = in[3];
	else
		out = in[4];
}
```

Mouse-over example:
The user mouses over the = on line 10.



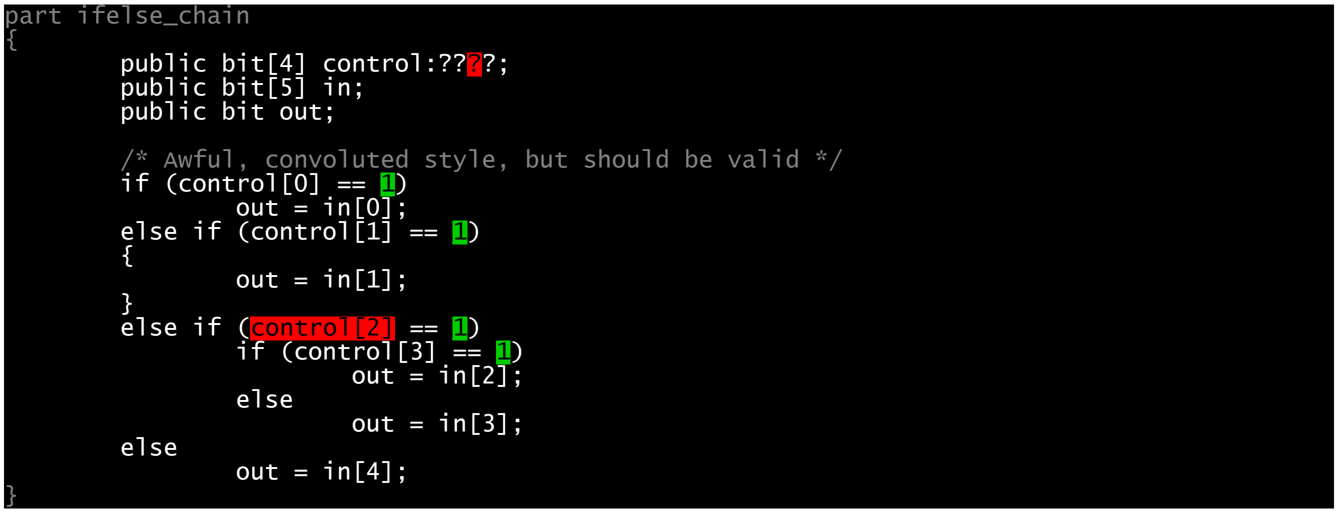
```
part ifelse_chain
{
        public bit[4] control:????;
        public bit[5] in;
        public bit out;

        /* Awful, convoluted style, but should be valid */
        if (control[0] == 1)
                out = in[0];
        else if (control[1] == 1)
        {
                out = in[1];
        }
        else if (control[2] == 1)
                if (control[3] == 1)
                        out = in[2];
                else
                        out = in[3];
        else
                out = in[4];
}
```

Mouse-over example:
The user hits v while over the 'control' declaration on line 3.



```
part ifelse_chain
{
        public bit[4] ????;
        public bit[5] in;
        public bit out;

        /* Awful, convoluted style, but should be valid */
        if (control[0] == 1)
                out = in[0];
        else if (control[1] == 1)
        {
                out = in[1];
        }
        else if (control[2] == 1)
                if (control[3] == 1)
                        out = in[2];
                else
                        out = in[3];
        else
                out = in[4];
}
```

Mouse-over example:
The user hits v a second time.

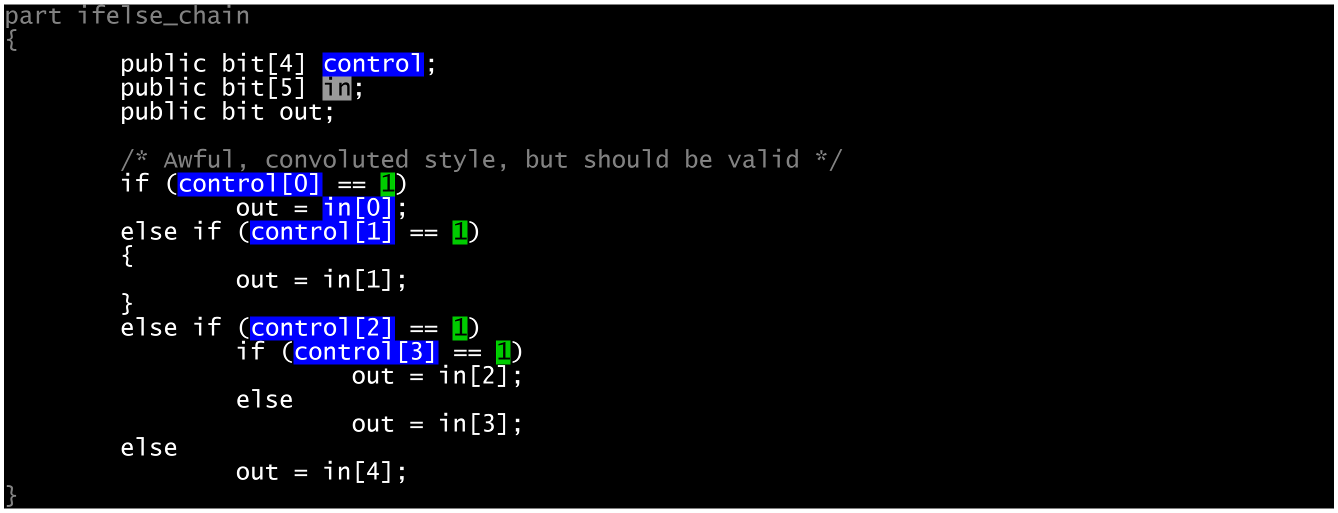


```
part ifelse_chain
{
        public bit[4] control:????;
        public bit[5] in;
        public bit out;

        /* Awful, convoluted style, but should be valid */
        if (control[0] == 1)
                out = in[0];
        else if (control[1] == 1)
        {
                out = in[1];
        }
        else if (control[2] == 1)
                if (control[3] == 1)
                        out = in[2];
                else
                        out = in[3];
        else
                out = in[4];
}
```

Mouse-over example:
The user returns control to “name:value” state, and then mouses over the 3rd ?.

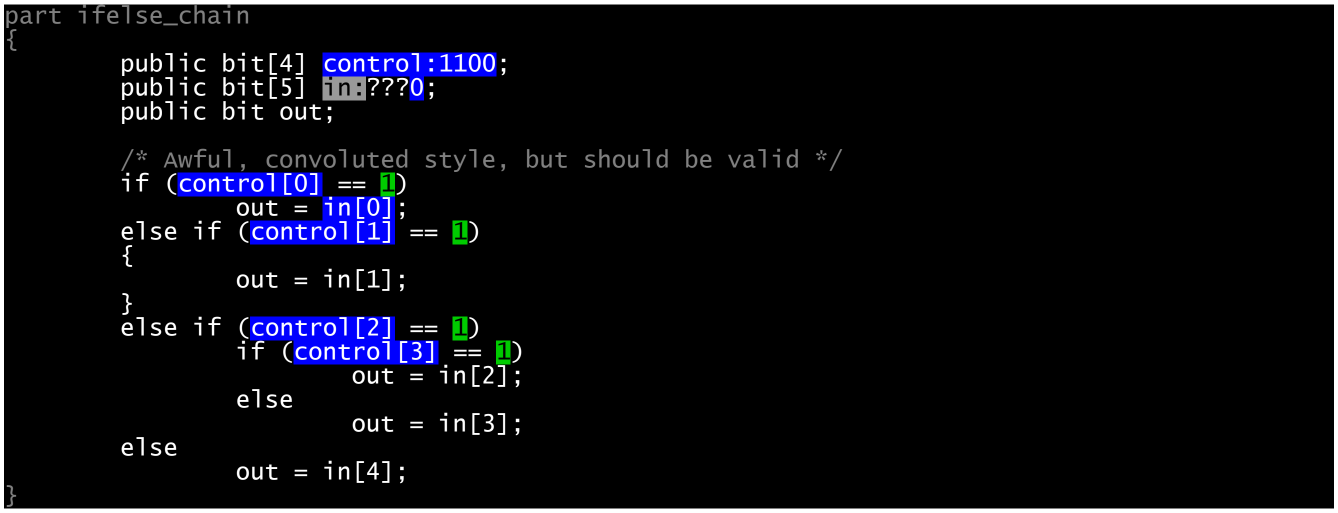
```
part ifelse_chain
{
	public bit[4] control;
	public bit[5] in;
	public bit out;

	/* Awful, convoluted style, but should be valid */
	if (control[0] == 1)
		out = in[0];
	else if (control[1] == 1)
	{
		out = in[1];
	}
	else if (control[2] == 1)
		if (control[3] == 1)
			out = in[2];
		else
			out = in[3];
	else
		out = in[4];
}
```

Mouse-over example:
The user moves the cursor away, and then hits 'space'. 'control' input gets all of its value set.



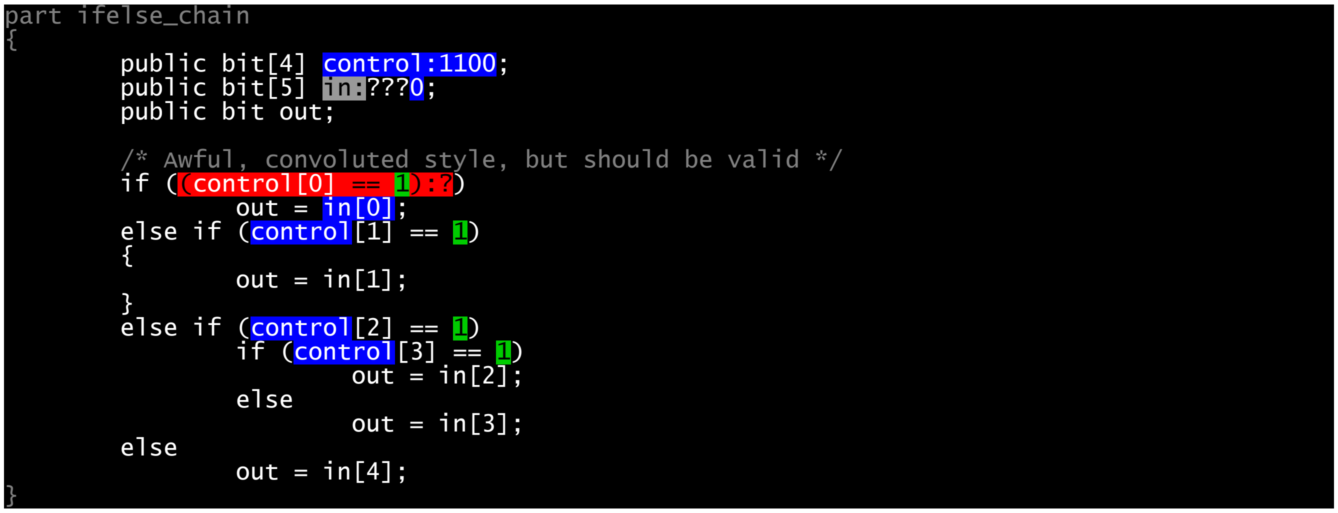
```
part ifelse_chain
{
        public bit[4] control;
        public bit[5] in;
        public bit out;

        /* Awful, convoluted style, but should be valid */
        if (control[0] == 1)
                out = in[0];
        else if (control[1] == 1)
        {
                out = in[1];
        }
        else if (control[2] == 1)
                if (control[3] == 1)
                        out = in[2];
                else
                        out = in[3];
        else
                out = in[4];
}
```

Mouse-over example:
The user hits 'space' again. 'in[0]' gets its value set, but the rest of the array is not set yet.



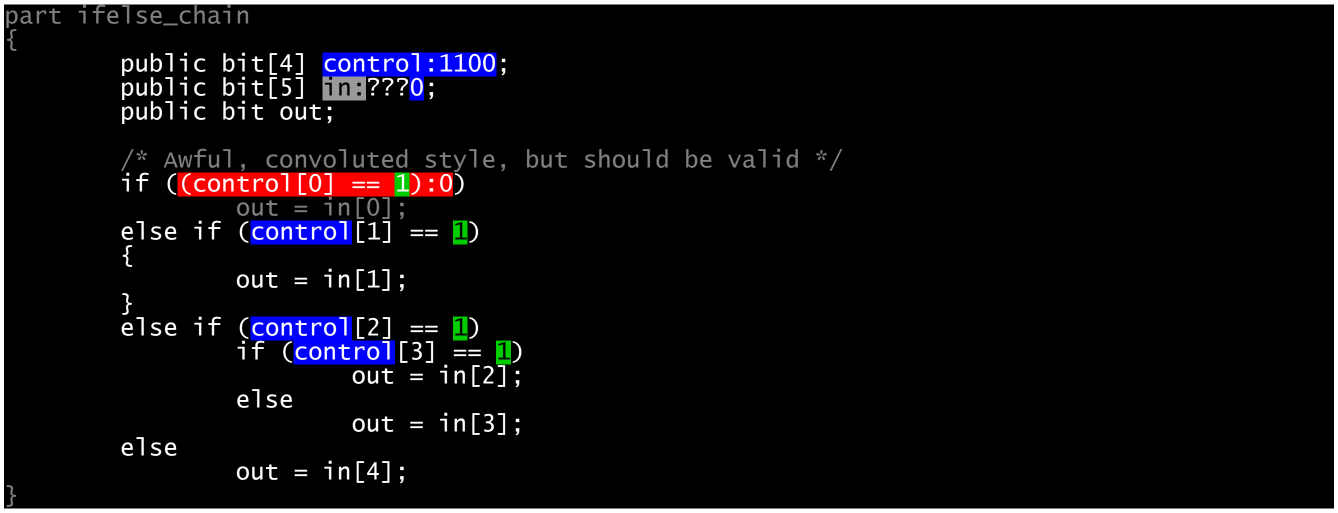
```
part ifelse_chain
{
	public bit[4] control:1100;
	public bit[5] in:???0;
	public bit out;

	/* Awful, convoluted style, but should be valid */
	if (control[0] == 1)
		out = in[0];
	else if (control[1] == 1)
	{
		out = in[1];
	}
	else if (control[2] == 1)
		if (control[3] == 1)
			out = in[2];
		else
			out = in[3];
	else
		out = in[4];
}
```

Mouse-over example:
The user turns on value display of both 'control' and 'in'.



```
part ifelse_chain
{
        public bit[4] control:1100;
        public bit[5] in:???0;
        public bit out;

        /* Awful, convoluted style, but should be valid */
        if ((control[0] == 1):?)
                out = in[0];
        else if (control[1] == 1)
        {
                out = in[1];
        }
        else if (control[2] == 1)
                if (control[3] == 1)
                        out = in[2];
                else
                        out = in[3];
        else
                out = in[4];
}
```

Mouse-over example:
The user mouses over the == on line 8, and hits v.

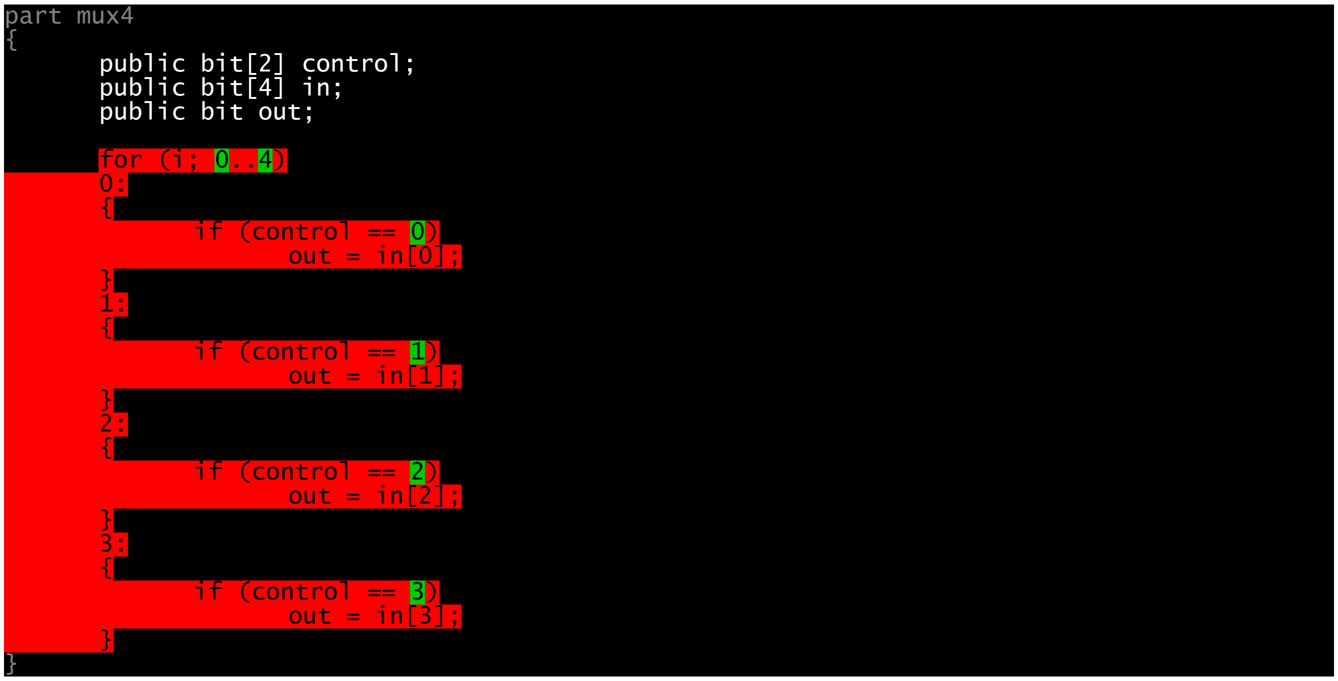


```
part ifelse_chain
{
        public bit[4] control:1100;
        public bit[5] in:???0;
        public bit out;

        /* Awful, convoluted style, but should be valid */
        if ((control[0] == 1):0)
                out = in[0];
        else if (control[1] == 1)
        {
                out = in[1];
        }
        else if (control[2] == 1)
                if (control[3] == 1)
                        out = in[2];
                else
                        out = in[3];
        else
                out = in[4];
}
```

Mouse-over example:
The user stays over the == on line 8, and hits ENTER. Notice that line 9 gets grayed out.

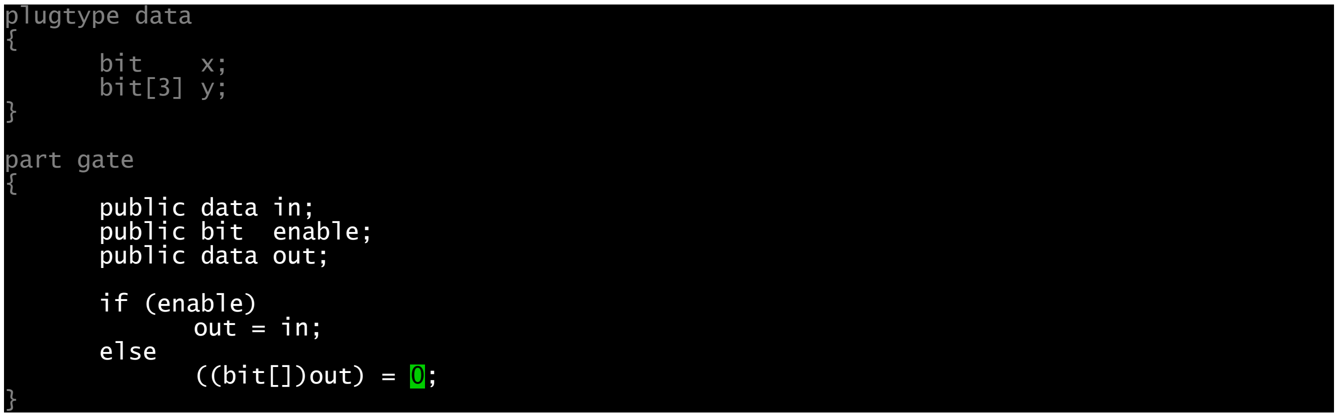
```
part mux4
{
      public bit[2] control;
      public bit[4] in;
      public bit out;

      for (i; 0..4)
      {
            if (control == i)
                  out = in[i];
      }
}
```

Mouse-over example:
The starting state. Note that 'i' is green (indicating that it is a compile-time constant). Technically, the array index is also the same constant, but we do **NOT** mark array indices as green.



```
part mux4
{
	public bit[2] control;
	public bit[4] in;
	public bit out;

	for (i; 0..4)
	0:
	{
		if (control == 0)
			out = in[0];
	}
	1:
	{
		if (control == 1)
			out = in[1];
	}
	2:
	{
		if (control == 2)
			out = in[2];
	}
	3:
	{
		if (control == 3)
			out = in[3];
	}
}
```

Mouse-over example:
User mouses over the “for” keyword and hits x.

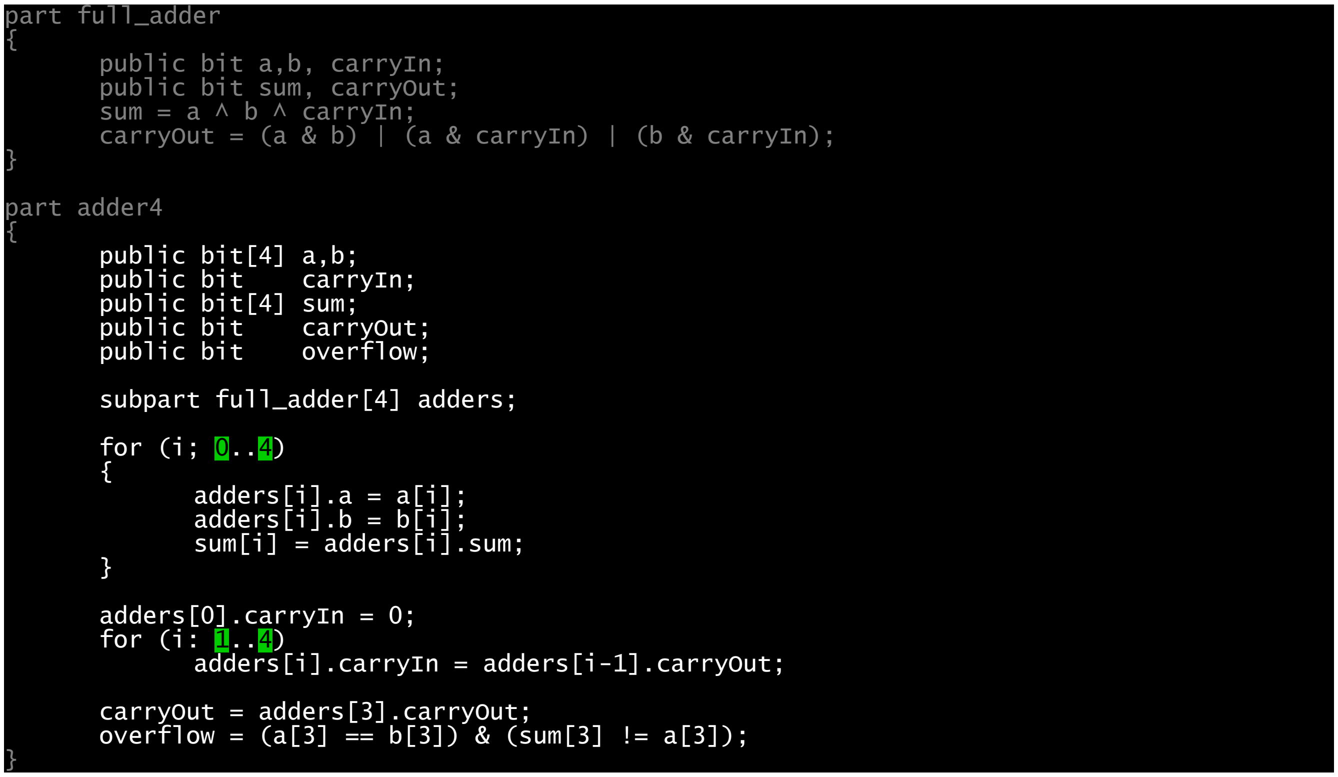


```
plugtype data
{
      bit    x;
      bit[3] y;
}

part gate
{
      public data in;
      public bit  enable;
      public data out;

      if (enable)
            out = in;
      else
            ((bit[])out) = 0;
}
```

The starting state.

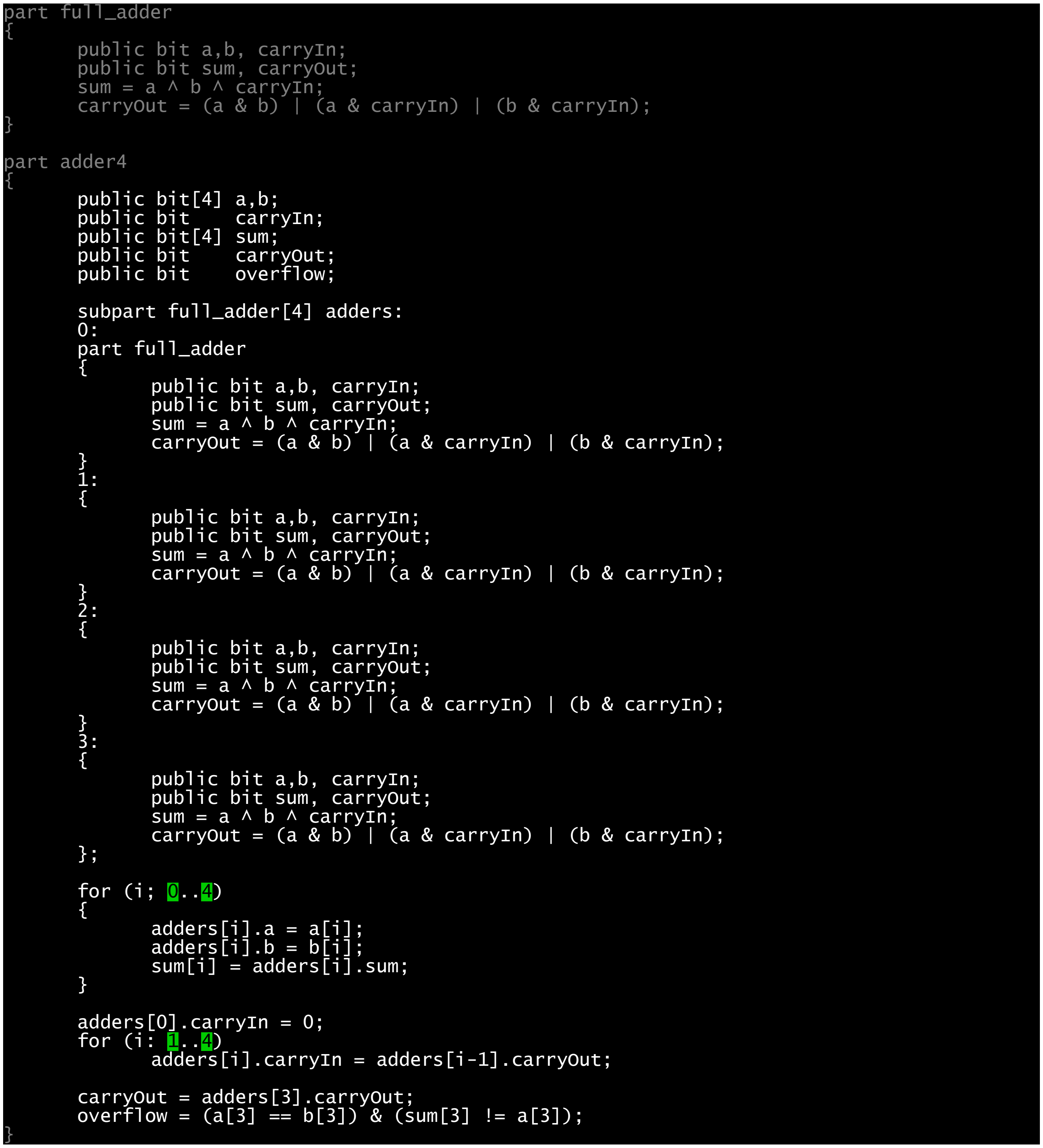
```
plugtype data
{
      bit    x;
      bit[3] y;
}

part gate
{
      public data in:{
                         bit    x;
                         bit[3] y;
                     };
      public bit  enable;
      public data out;

      if (enable)
            out = in;
      else
            ((bit[])out) = 0;
}
```

The mouses over the 'in' declaration and hits x. The definition of the 'data' plugtype is dropped into the code. The various fields and code of the plugtype can now be accessed the same was as the “direct” fields and code of the enclosing part.



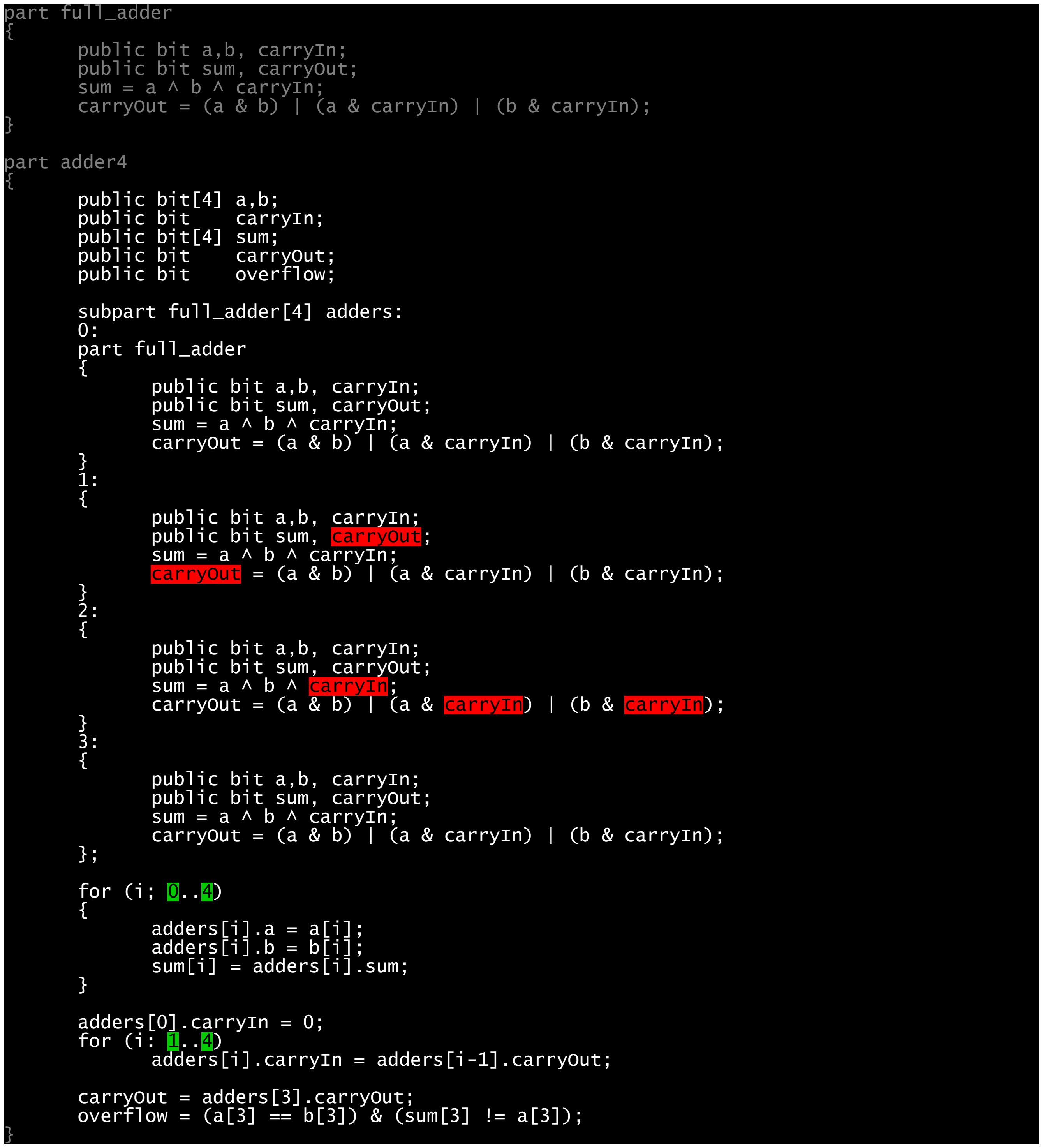
```
part full_adder
{
      public bit a,b, carryIn;
      public bit sum, carryOut;
      sum = a ^ b ^ carryIn;
      carryOut = (a & b) | (a & carryIn) | (b & carryIn);
}

part adder4
{
      public bit[4] a,b;
      public bit    carryIn;
      public bit[4] sum;
      public bit    carryOut;
      public bit    overflow;

      subpart full_adder[4] adders;

      for (i; 0..4)
      {
            adders[i].a = a[i];
            adders[i].b = b[i];
            sum[i] = adders[i].sum;
      }

      adders[0].carryIn = 0;
      for (i: 1..4)
            adders[i].carryIn = adders[i-1].carryOut;

      carryOut = adders[3].carryOut;
      overflow = (a[3] == b[3]) & (sum[3] != a[3]);
}
```

A more complex part, including 4 sub-parts.



```
part full_adder
{
	public bit a,b, carryIn;
	public bit sum, carryOut;
	sum = a ^ b ^ carryIn;
	carryOut = (a & b) | (a & carryIn) | (b & carryIn);
}

part adder4
{
	public bit[4] a,b;
	public bit    carryIn;
	public bit[4] sum;
	public bit    carryOut;
	public bit    overflow;

	subpart full_adder[4] adders:
	0:
	part full_adder
	{
		public bit a,b, carryIn;
		public bit sum, carryOut;
		sum = a ^ b ^ carryIn;
		carryOut = (a & b) | (a & carryIn) | (b & carryIn);
	}
	1:
	{
		public bit a,b, carryIn;
		public bit sum, carryOut;
		sum = a ^ b ^ carryIn;
		carryOut = (a & b) | (a & carryIn) | (b & carryIn);
	}
	2:
	{
		public bit a,b, carryIn;
		public bit sum, carryOut;
		sum = a ^ b ^ carryIn;
		carryOut = (a & b) | (a & carryIn) | (b & carryIn);
	}
	3:
	{
		public bit a,b, carryIn;
		public bit sum, carryOut;
		sum = a ^ b ^ carryIn;
		carryOut = (a & b) | (a & carryIn) | (b & carryIn);
	};

	for (i; 0..4)
	{
		adders[i].a = a[i];
		adders[i].b = b[i];
		sum[i] = adders[i].sum;
	}

	adders[0].carryIn = 0;
	for (i: 1..4)
		adders[i].carryIn = adders[i-1].carryOut;

	carryOut = adders[3].carryOut;
	overflow = (a[3] == b[3]) & (sum[3] != a[3]);
}
```

The user hits x on the adders declaration. (Note that the picture above assumes that the user is no longer over the top of the declaration, so nothing is highlighted.)
]



```
part full_adder
{
        public bit a,b, carryIn;
        public bit sum, carryOut;
        sum = a ^ b ^ carryIn;
        carryOut = (a & b) | (a & carryIn) | (b & carryIn);
}

part adder4
{
        public bit[4] a,b;
        public bit    carryIn;
        public bit[4] sum;
        public bit    carryOut;
        public bit    overflow;

        subpart full_adder[4] adders:
        0:
        part full_adder
        {
                public bit a,b, carryIn;
                public bit sum, carryOut;
                sum = a ^ b ^ carryIn;
                carryOut = (a & b) | (a & carryIn) | (b & carryIn);
        }
        1:
        {
                public bit a,b, carryIn;
                public bit sum, carryOut;
                sum = a ^ b ^ carryIn;
                carryOut = (a & b) | (a & carryIn) | (b & carryIn);
        }
        2:
        {
                public bit a,b, carryIn;
                public bit sum, carryOut;
                sum = a ^ b ^ carryIn;
                carryOut = (a & b) | (a & carryIn) | (b & carryIn);
        }
        3:
        {
                public bit a,b, carryIn;
                public bit sum, carryOut;
                sum = a ^ b ^ carryIn;
                carryOut = (a & b) | (a & carryIn) | (b & carryIn);
        };

        for (i; 0..4)
        {
                adders[i].a = a[i];
                adders[i].b = b[i];
                sum[i] = adders[i].sum;
        }

        adders[0].carryIn = 0;
        for (i: 1..4)
                adders[i].carryIn = adders[i-1].carryOut;

        carryOut = adders[3].carryOut;
        overflow = (a[3] == b[3]) & (sum[3] != a[3]);
}
```

When the user mouses over a field inside one of the sub-parts, it also highlights the fields in other sub-parts to which it is directly connected. If applicable, it **also** connects to fields in the enclosing part.

QUESTION: Should we include a “partial highlight” option, to highlight terms inside for() loops which represent these highlighted terms in some but not all iterations of the loop?

QUESTION: If we mouse over a term in a loop (when the loop is *NOT* expanded), should we figure out what all of the instances of the term would be (when the loop *WAS* expanded), along with the various things that they are connected to?