**Keygen.v**

This Verilog code is for a module called `keygen` which generates a key for a cryptographic system.

The module takes four inputs: `p`, `q`, `start`, and `clk`, and two outputs: `e1` and `finish`.

`p` and `q` are 8-bit inputs used to calculate the value of `phin` which represents the **Euler totient** function of `n`, where `n = p \* q`.

`start` is a control input signal that initiates the key generation process.

`clk` is the clock input signal.

`e1` is an 8-bit output that represents the public key generated by the module.

`finish` is a 1-bit output that signals the end of the key generation process.

The key generation process begins when `start` is asserted. The module then calculates the value of `phin` and initializes the values of `x`, `y`, `random`, `gcd`, and `fin`.

The module then starts searching for a suitable value of `e` that is coprime with `phin`. It does this by using the extended Euclidean algorithm to find the greatest common divisor (`gcd`) of `y` and `phin` until `gcd` is equal to 1. At each iteration, the module checks if `gcd` is equal to 1 and if so, sets the output `e1` to the current value of `random` and sets `finish` to 1 to indicate that the key generation process is complete. If `gcd` is not equal to 1, the module updates the value of `random` to the next odd number and starts the search process again.

The module uses a custom Verilog module called `Divider` to perform the division operation needed in the extended Euclidean algorithm. The `Divider` module takes two inputs, `x` and `y`, and outputs the quotient `q` and the remainder `r` of `x` divided by `y`.

**Divider.v and Divider32.v:**

This is a Verilog module that implements a hardware divider for two input operands A and B. The result of the division is stored in the output port Res, and the remainder is stored in the output port remainder. The module takes in two input ports A and B, each of width WIDTH (default value 16). The output ports Res and remainder are also of width WIDTH.

The module uses a **simple non-restoring division algorithm** to perform the division. It uses a for loop to iterate through each bit of the input operands A and B. The variables a1 and b1 are used to hold the current value of A and B, respectively. The variable p1 holds the current value of the partial product (initially set to 0).Inside the for loop, the partial product p1 is shifted left by one bit and the most significant bit of a1 is inserted into the least significant bit of p1. The value of a1 is also shifted left by one bit to prepare for the next iteration.

Then, the partial product p1 is subtracted by B. If the sign bit (most significant bit) of p1 is 1, it means that the result of the subtraction is negative, so the value of the least significant bit of a1 (which is equal to the remainder of the division at this stage) is set to 0, and the value of p1 is added with B to restore it. Otherwise, the least significant bit of a1 is set to 1.

After the for loop completes, the value of a1 represents the quotient of the division and is stored in the output port Res. The value of p1 is equal to the remainder of the division and is split into two parts, remainderL (lower 8 bits) and remainder (upper 8 bits), and stored in the output port remainder.

Note that there is a naming conflict in the module where the output port Res is declared twice, once as an output port and again as a reg variable. This can cause errors during compilation. It is recommended to remove the redundant declaration of the Res variable in the module.

**Dnew.v:**

This is a Verilog code for implementing a module that performs modular inverse calculation using the **Extended Euclidean Algorithm.** The module takes four input signals, which are:

- p: an 8-bit input signal representing the value of p in the equation **(p\*q) mod e = 1.**

- q: an 8-bit input signal representing the value of q in the equation (p\*q) mod e = 1.

- e1: an 8-bit input signal representing the value of e in the equation (p\*q) mod e = 1.

- clk: a clock input signal.

The module has three output signals, which are:

- n: a 16-bit output signal representing the value of (p\*q) in the equation (p\*q) mod e = 1.

- d: a 16-bit output signal representing the calculated modular inverse value.

- finished: an output signal that goes high when the calculation is finished.

The code defines a few registers and wires used in the algorithm, including A, B, C, G, and e. The algorithm starts when the start signal goes high. At that point, the module sets the values of e, n, G, A, and B. Then, it enters a loop where it performs the Extended Euclidean Algorithm until B[15:0] equals 1.

In each iteration of the loop, the module calculates the quotient **Q=A[15:0]/B[15:0] and the remainder C=A-Q\*B.** It then updates the values of A and B to continue the algorithm. Finally, the module sets the value of d to be B[31:16] and the finished signal to be high when B[15:0] equals 1.

Note that the code uses a module called Divider, which is not defined in the code provided. The Divider module is likely used to calculate the quotient and remainder values needed for the algorithm.

**Modularmult.v:**

This Verilog code defines a module for performing modular multiplication using the **Montgomery algorithm.** The module has several inputs and outputs, as follows:

Inputs:

- `M`: a 16-bit input representing the base value of the modular multiplication

- `e`: a 16-bit input representing the exponent value of the modular multiplication

- `n`: a 16-bit input representing the modulus value of the modular multiplication

- `start`: a 1-bit input representing the start signal for beginning the modular multiplication

- `clk`: a 1-bit input representing the clock signal for the module

Outputs:

- `finished`: a 1-bit output representing the finished signal for indicating when the modular multiplication has completed

- `Mpower`: a 32-bit output representing the result of the modular multiplication

- `remainder`: a 16-bit output representing the remainder value during the modular multiplication process

The module starts by instantiating a `Divider32` module to perform division of the intermediate values during the modular multiplication process. The `Divider32` module takes in `x` and `n1` as inputs, and produces `outResult` and `remainder` as outputs.

The module also contains an `always` block that is triggered on the positive edge of the clock signal. If the `start` signal is asserted, the module initializes the `ncount` register to the value of `e-1`, sets the initial value of `Mpower` to `M`, sets `x` to 0, and sets `n1` to a concatenation of 16 zeros and the value of `n`. If the `finished` signal is not yet asserted, the module multiplies the current `remainder` value with `M` to obtain the updated value of `Mpower`, and decrements the value of `ncount` by 1. Finally, the updated value of `Mpower` is assigned to `x`.

The `assign` statement at the end of the module sets the value of `finished` to 1 if `ncount` is equal to 0, indicating that the modular multiplication process has completed.

**Main.v:**

This is a Verilog module called "Main", which has several inputs and outputs. Here is a brief explanation of each input/output:

Inputs:

- M: a 16-bit input value

- p: an 8-bit input value used for key generation

- q: an 8-bit input value used for key generation

- clk: a clock signal input

- start: a control input signal used for key generation

- start1: a control input signal used for dnew module

- start2: a control input signal used for modularmult module

Outputs:

- e: an 8-bit output value generated by keygen module

- n: a 16-bit output value generated by dnew module

- remainder: a 16-bit output value generated by modularmult module

- d: a 16-bit output value generated by dnew module

- finish: a control output signal generated by keygen module

- fin1: a control output signal generated by dnew module

The module also instantiates three sub-modules:

- keygen: This module generates keys (e, d, and n) based on the input values p and q.

- dnew: This module generates a new d value and an n value based on the input values p, q, and e.

- modularmult: This module performs modular multiplication of two input values (M and d) and generates a remainder.

There are also some commented out code sections that include an input/output selection logic using case statements. However, these are not currently used in the module.

**Testbench:**

This is a Verilog code for a test bench module, which is used to verify the functionality of another module called "Main". The test bench module is responsible for providing inputs to the Main module and verifying its outputs against expected values.

The inputs to the Main module are declared as registers in the test bench module, and the outputs are declared as wires. The UUT (Unit Under Test) instance is created using the Main module and its inputs and outputs are connected to the corresponding signals in the test bench module.

In the initial block, the inputs are initialized to their default values and then some values are assigned to M, p, and q. After a delay of 10 units of simulation time, the start signal is set to 1, indicating the start of the test. After another delay of 5 units, start signal is set back to 0. Then, after a delay of 40 units, start1 is set to 1 for 10 units and then set back to 0. Similarly, start2 is set to 1 after a delay of 30 units for 10 units and then set back to 0.

The always block toggles the value of the clk signal every 0.1 units of simulation time. This creates a clock signal that is used to synchronize the inputs and outputs of the Main module.

The test bench module can be simulated to verify the correctness of the Main module. Any errors or unexpected behaviors can be identified and fixed before the Main module is integrated into a larger system.