Presentation of 2nd assignment - ece437

Dimitrios Tsalapatas
03246
Algorithm cad 1

07/01/2023

Concept

The main concept of HW_2 is to implement a parser for practical format files.

Also, we need to store information about the design on hash tables.

After this we need to be able to print all necessary information using TCL command

Example of practical format file

```
############################################################
# Generated by: PathVIZ (Version: 1.0)
# OS: Linux x86_64(Host: haflife)
# Generated on: Sun Nov 19 09:29:14 2023
# Design: c17
############################################################
# Core Utilisation: 64%
# Core Width, Height: 7.000, 7.560, Aspect Ratio: 0.926
# Core X, Y Offsets: 0.000, 0.000
############################################################
# Rows:
Row: COREROW_0 Type: CoreSite Location: 0.000 0.000 Width/Height: 7.000 3.780
Row: COREROW_1 Type: CoreSite Location: 0.000 3.780 Width/Height: 7.000 3.780
############################################################
# Top-Level I/O Ports:
IO: N1 Location: 0.000 3.800
# WEST SIDE
IO: N2 Location: 3.430 7.560
# SOUTH SIDE
IO: N3 Location: 7.000 3.800
# EAST SIDE
IO: N6 Location: 3.430 0.000
# NORTH SIDE
IO: N7 Location: 0.000 3.900
# WEST SIDE
IO: clk Location: 3.570 7.560
# SOUTH SIDE
IO: N22 Location: 7.000 3.900
# EAST SIDE
IO: N23 Location: 3.570 0.000
# NORTH SIDE
############################################################
# Top-Level I/O CCs:
IO: N1 CCs: c17/g66__8428 (/A)
IO: N2 CCs: c17/g64__6260 (/A)
IO: N3 CCs: c17/g66__8428 (/B) c17/g67__5526 (/A)
IO: N6 CCs: c17/g67__5526 (/B)
IO: N7 CCs: c17/g65__4319 (/B)
IO: clk CCs:
IO: N22 CCs: c17/g62__2398 (/Q)
IO: N23 CCs: c17/g63__5107 (/Q)
############################################################
# Components CCs:
Component: c17/g62__2398 Cell_Type: NA2JILTX1 Cell_Timing_Type: Combinational Width: 1.500 Height: 3.780 CCs: c17/g62__2398 (/Q)
Component: c17/g62__2398, Output Pin: Q, Boolean Function: !(A*B)
Component: c17/g63__5107 Cell_Type: NA2JILTX1 Cell_Timing_Type: Combinational Width: 1.500 Height: 3.780 CCs: c17/g63__5107 (/Q)
Component: c17/g63__5107, Output Pin: Q, Boolean Function: !(A*B)
Component: c17/g64__6260 Cell_Type: NA2JILTX1 Cell_Timing_Type: Combinational Width: 1.500 Height: 3.780 CCs: c17/g64__6260 (/Q) c17/g62__2398 (/A) c17/g63__5107 (/A)
Component: c17/g64__6260, Output Pin: Q, Boolean Function: !(A*B)
Component: c17/g67__5526 Cell_Type: NA2JILTX1 Cell_Timing_Type: Combinational Width: 1.500 Height: 3.780 CCs: c17/g67__5526 (/Q) c17/g64__6260 (/A) c17/g65__4319 (/A)
Component: c17/g67__5526, Output Pin: Q, Boolean Function: !(A*B)
Component: c17/g65__4319 Cell_Type: NA2JILTX1 Cell_Timing_Type: Combinational Width: 1.500 Height: 3.780 CCs: c17/g65__4319 (/Q) c17/g63__5107 (/B)
Component: c17/g65__4319, Output Pin: Q, Boolean Function: !(A*B)
Component: c17/g66__8428 Cell_Type: NA2JILTX1 Cell_Timing_Type: Combinational Width: 1.500 Height: 3.780 CCs: c17/g66__8428 (/Q) c17/g62__2398 (/B)
Component: c17/g66__8428, Output Pin: Q, Boolean Function: !(A*B)
```

Structs

There are three main hash tables:

- Comphash: stores informations about each component
- Gatepinhash: stores informations about each gatepin
- Libhash: stores informations about each different cell

Each hashtable has a static HASHDEPTH that we define it as 20
If it has more than 20 collisions program prints error and exits

Gatepins Hashtable

gatepinhash:

- Char *name[HASHDEPTH]: name of pin
- Int *pinConn[HASHDEPTH]: array with hash values of successors of this pin
- Int *pinConnDepth[HASHDEPTH]: array with hash depth of successors of this pin
- Int connections_size[HASHDEPTH]: number of connections for each pin
- Int parentComponent[HASHDEPTH]: hash value for component
- Int parentComponentDepth[HASHDEPTH]: hash depth for component
- Int type[HASHDEPTH]: Wire or IO
- Int hashpresent[HASHDEPTH]: variable to show us if bucket is empty or not

```
struct gatepins
{
    char *name[HASHDEPTH];

    // connections //
    int *pinConn[HASHDEPTH]; // hash key //
    int *pinConnDepth[HASHDEPTH];  // depth t
    int connections_size[HASHDEPTH];

    // parent component //
    int parentComponent[HASHDEPTH];
    int parentComponentDepth[HASHDEPTH];
    int type[HASHDEPTH];   /* what type this

    int hashpresent[HASHDEPTH];
```

Components Hashtable

Comphash:

- Char *name[HASHDEPTH]: name of component
- Int lib_type[HASHDEPTH]: array with hash values of cell type
- Int lib_type_depth[HASHDEPTH]: array with hash depth of cell type
- Int hashpresent[HASHDEPTH]: variable to show us if bucket is empty or not

```
struct components
{
    char *name[HASHDEPTH];
    int lib_type[HASHDEPTH];
    int lib_type_depth[HASHDEPTH];
    int hashpresent[HASHDEPTH];

};
```

Library Hashtable

libhash:

- Char *name[HASHDEPTH]: name of cell
- Int cell_type[HASHDEPTH]: type of each cell (Combinational or Sequential)
- Int *pin_names[HASHDEPTH]: names of pins for each cell
- Int pin_count[HASHDEPTH]: number of pins current cell has
- Char **function[HASHDEPTH]: boolean function for each cell (it can have more than one outputs)
- Int out_pins_count[HASHDEPTH]: number of output pins
- Int *pin_type[HASHDEPTH]: Input or Output
- Int hashpresent[HASHDEPTH]: variable to show us if bucket is empty or not

```
struct library
{
    char *name[HASHDEPTH]; // name
    int cell_type[HASHDEPTH]; // t
    char **pin_names[HASHDEPTH]; /
    int pin_count[HASHDEPTH]; // c
    char **function[HASHDEPTH]; //
    int out_pins_count[HASHDEPTH];
    int hashpresent[HASHDEPTH]; //
    int *pin_type[HASHDEPTH];
};
```

Parser

```
############################################################
# Generated by: PathVIZ (Version: 1.0)
# OS: Linux x86_64(Host: haflife)
# Generated on: Sun Nov 19 09:29:14 2023
# Design: c17
############################################################
# Core Utilisation: 64%
# Core Width, Height: 7.000, 7.560, Aspect Ratio: 0.926
# Core X, Y Offsets: 0.000, 0.000
############################################################
# Rows:
Row: COREROW_0 Type: CoreSite Location: 0.000 0.000 Width/Height: 7.000 3.780
Row: COREROW_1 Type: CoreSite Location: 0.000 3.780 Width/Height: 7.000 3.780
############################################################
# Top-Level I/O Ports:
IO: N1 Location: 0.000 3.800
# WEST SIDE
IO: N2 Location: 3.430 7.560
# SOUTH SIDE
IO: N3 Location: 7.000 3.800
# EAST SIDE
IO: N6 Location: 3.430 0.000
# NORTH SIDE
IO: N7 Location: 0.000 3.900
# WEST SIDE
IO: clk Location: 3.570 7.560
# SOUTH SIDE
IO: N22 Location: 7.000 3.900
# EAST SIDE
IO: N23 Location: 3.570 0.000
# NORTH SIDE
############################################################
# Top-Level I/O CCs:
IO: N1 CCs: c17/g66__8428 (/A)
IO: N2 CCs: c17/g64__6260 (/B)
IO: N3 CCs: c17/g66__8428 (/B) c17/g67__5526 (/A)
IO: N6 CCs: c17/g67__5526 (/B)
IO: N7 CCs: c17/g65__4319 (/B)
IO: clk CCs:
IO: N22 CCs: c17/g62__2398 (/Q)
IO: N23 CCs: c17/g63__5107 (/Q)
############################################################
# Components CCs:
Component: c17/g62__2398 Cell_Type: NA2JILTX1 Cell_Timing_Type: Combinational Width: 1.500 Height: 3.780 CCs: c17/g62__2398 (/Q)
Component: c17/g62__2398, Output Pin: Q, Boolean Function: !(A*B)
Component: c17/g63__5107 Cell_Type: NA2JILTX1 Cell_Timing_Type: Combinational Width: 1.500 Height: 3.780 CCs: c17/g63__5107 (/Q)
Component: c17/g63__5107, Output Pin: Q, Boolean Function: !(A*B)
Component: c17/g64__6260 Cell_Type: NA2JILTX1 Cell_Timing_Type: Combinational Width: 1.500 Height: 3.780 CCs: c17/g64__6260 (/Q) c17/g62__2398 (/A) c17/g63__5107 (/A)
Component: c17/g64__6260, Output Pin: Q, Boolean Function: !(A*B)
Component: c17/g67__5526 Cell_Type: NA2JILTX1 Cell_Timing_Type: Combinational Width: 1.500 Height: 3.780 CCs: c17/g67__5526 (/Q) c17/g64__6260 (/A) c17/g65__4319 (/A)
Component: c17/g67__5526, Output Pin: Q, Boolean Function: !(A*B)
Component: c17/g65__4319 Cell_Type: NA2JILTX1 Cell_Timing_Type: Combinational Width: 1.500 Height: 3.780 CCs: c17/g65__4319 (/Q) c17/g63__5107 (/B)
Component: c17/g65__4319, Output Pin: Q, Boolean Function: !(A*B)
Component: c17/g66__8428 Cell_Type: NA2JILTX1 Cell_Timing_Type: Combinational Width: 1.500 Height: 3.780 CCs: c17/g66__8428 (/Q) c17/g62__2398 (/B)
Component: c17/g66__8428, Output Pin: Q, Boolean Function: !(A*B)
```

1

2

3

I split the file in 3 parts:

- 1st one has the names of IOs
- 2nd has the connections of each IO
- 3rd one contains all components cells and gatepins connections

Parser

Parser reads file line by line and search for specific string that indicates a different part of file. Based on this sentence it chooses the corresponding FSM.
After this it splits the line into words and pass them to FSM

```
// read first section of file line by line until it ends //
while((read_length = (getline(&line, &line_size, filename) ) )!= -1)
{
    j = 0;
    find_line = strstr(line, "Top-Level I/O CCs:");
    if(find_line != NULL)
    {
        choose_FSM = 1;  // set the choose_FSM that the following line has IO //
        correct_format = 1;
    }

    find_line = strstr(line, "Components CCs:");
    if(find_line != NULL)
    {
        choose_FSM = 3;  // set the choose_FSM that the following line has IO //
    }

    while(j < read_length) // proccess line to split it in words //
    {
        for(i = j; i < read_length+1; i++)
        {
            if(line[i] == ' ' || line[i] == '\0' || line[i] == '\v' || line[i] == '\r')
            {
                word[pos] = '\0';
                break;
            }
            else
            {
                word[pos] = line[i];
                pos++ ;
            }
        }
        pos = 0;
        if(choose_FSM == 1)
        {
            currentState2 = countIOS(currentState2, word);
        }
        else if (choose_FSM == 3)
        {
            currentState2 = count_components_CCS(currentState2, word);
        }

        while (line[i] == ' ')
        {
            i++;
        }
        j = i;
    }
    free(line);
    line = NULL;
}
```

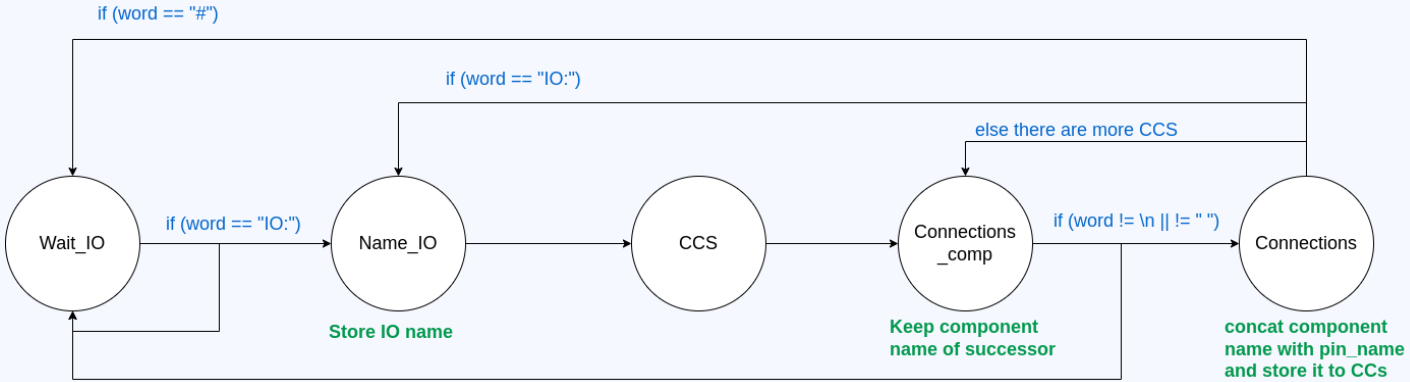
Parser

- In first parse of file we count number of Components and Gateping we find to initialize the hashtables.
  For cells we store them temporarily in a dynamic array and then find the exact number
  - Second parse: FSM that proccess IOS_CCs. It parses the 2nd part of file and store all IOs in gatepinhash and add all connections

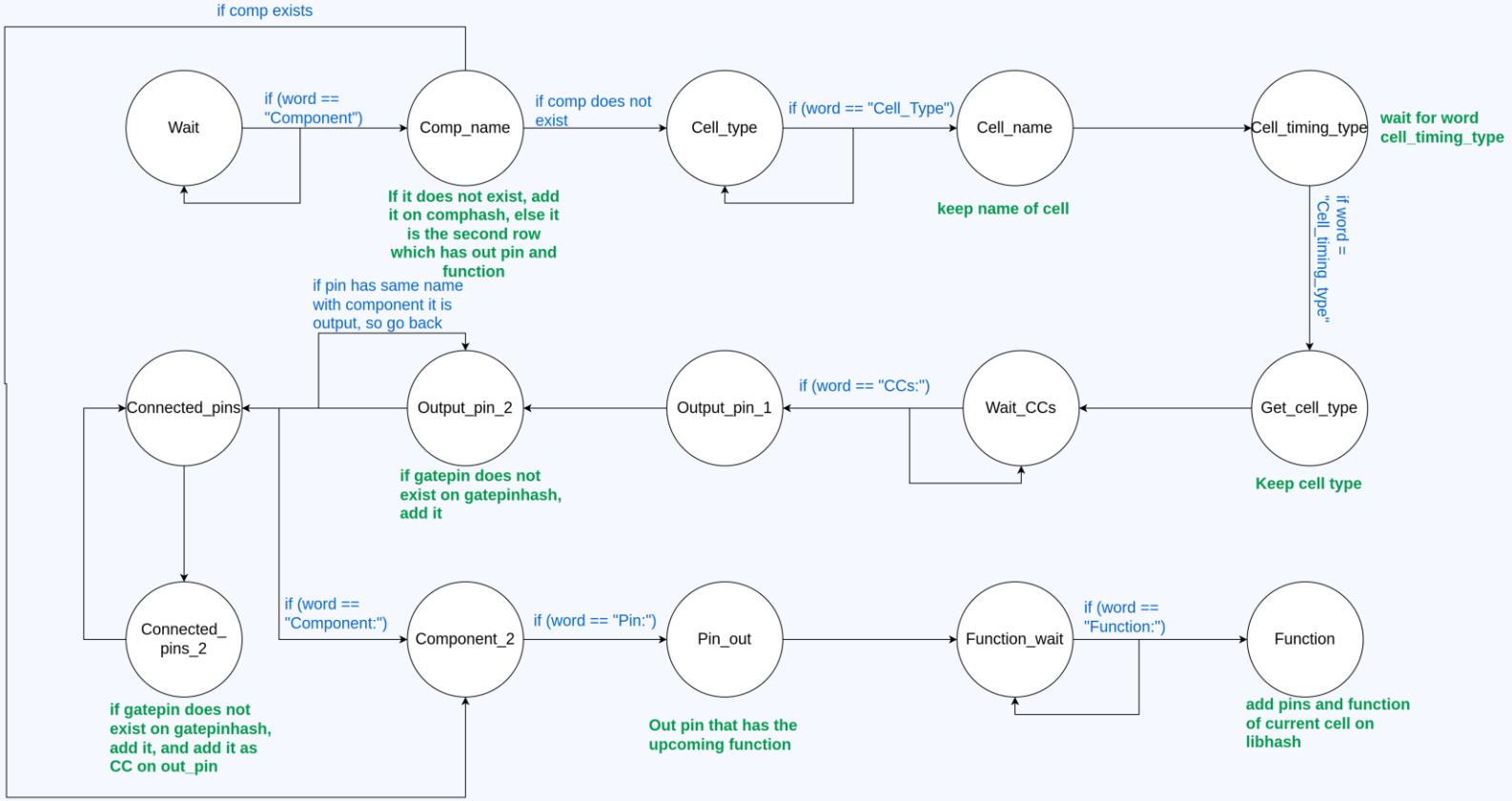
- For all FSMs we use enums to be easier to read states

```
enum IO_STATES_CCS
{
    WAIT_IO,
    NAME_IO,
    CCS,
    CONNECTIONS_COMP,
    CONNECTIONS
};
```

Example: IO: o_wb_data[0] CCs: PID/U70 (/Q)
IO: o_un[31] CCs: PID/\un_reg[31] (/Q) PID/U830 (/IN0) PID/U1263 (/IN0)



- Second parse: FSM that proccess components cells and gatepins. It parses the 3rd part of file and store gatepins with connections, components with cell type and cells

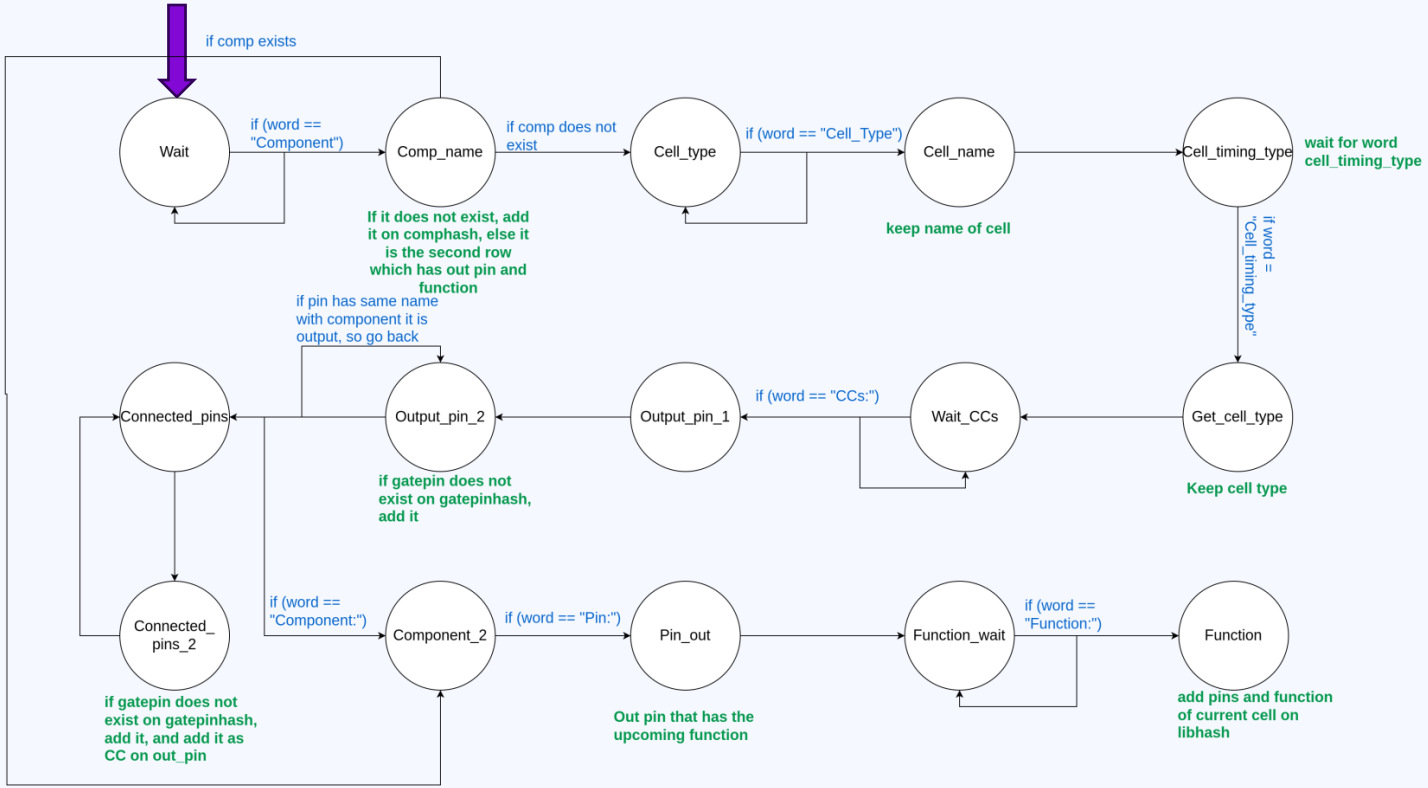


Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Component

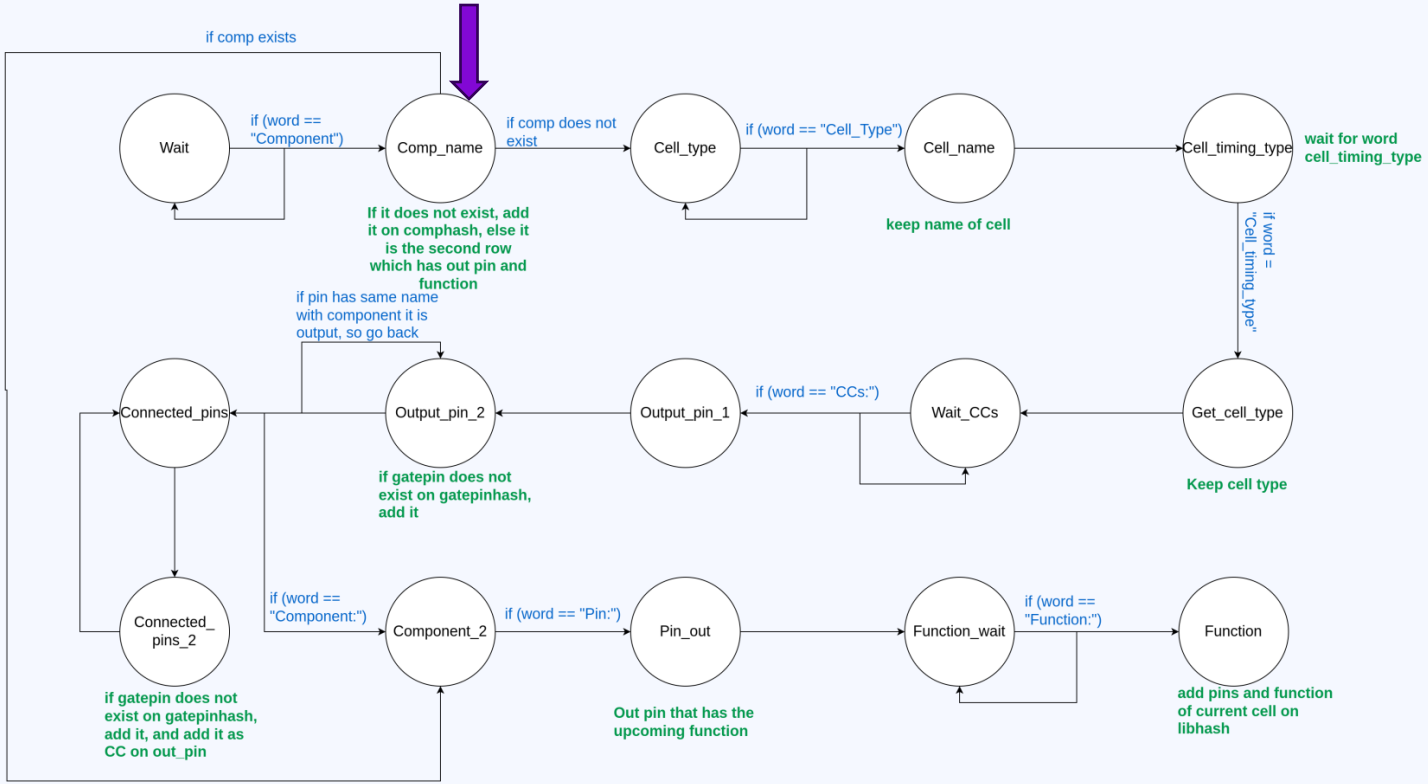


Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: three_counters/U29



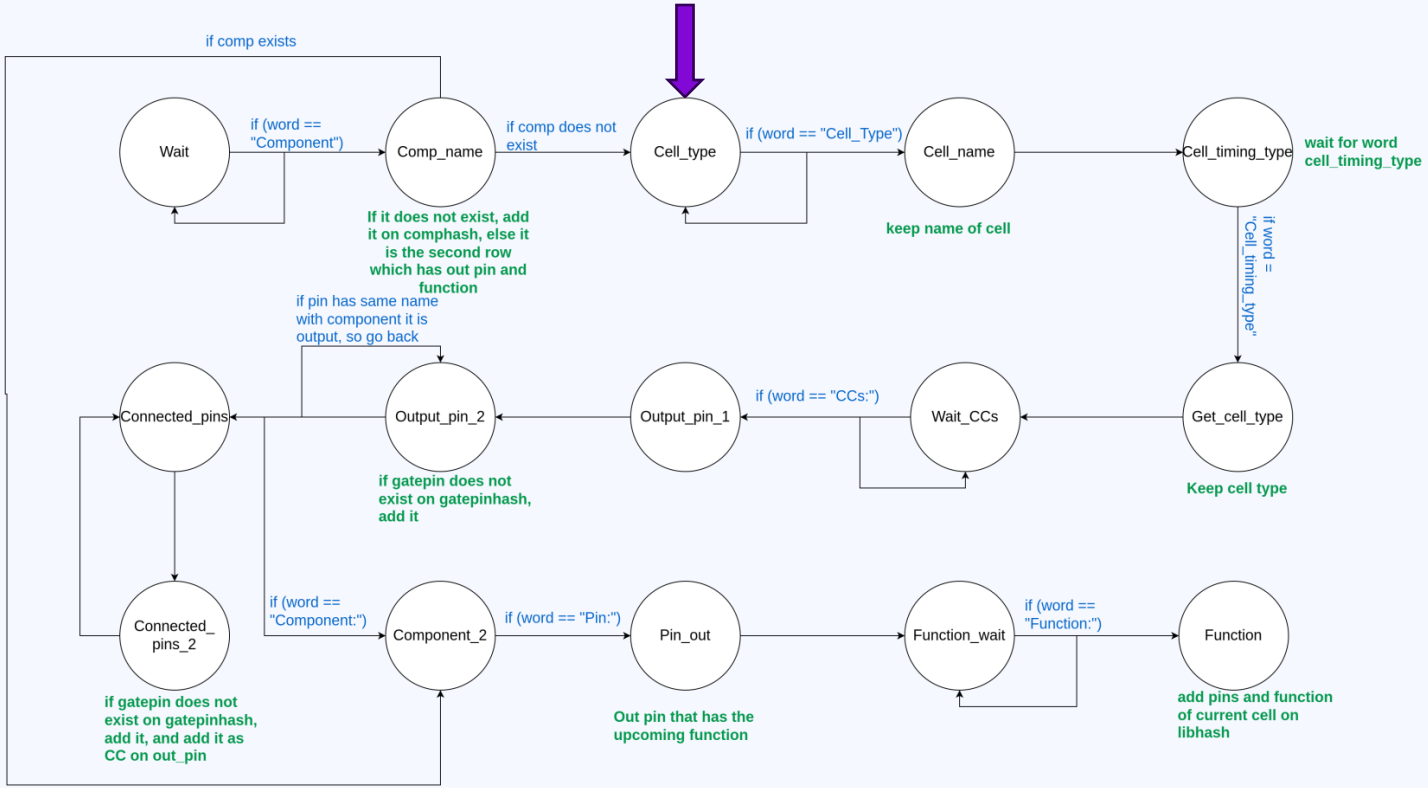
Comp_name

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Cell_Type



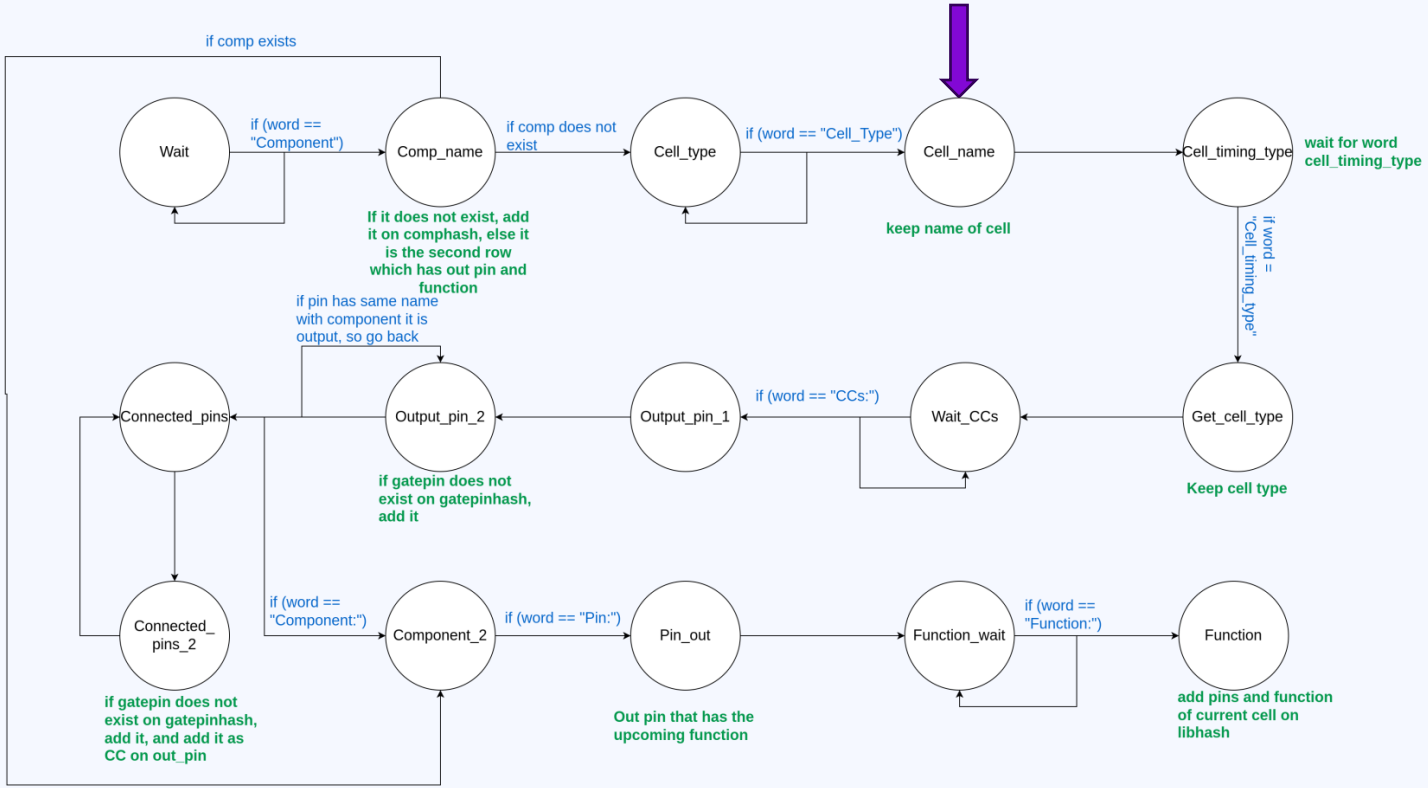
Comp_name

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: add2d1



Comp_name

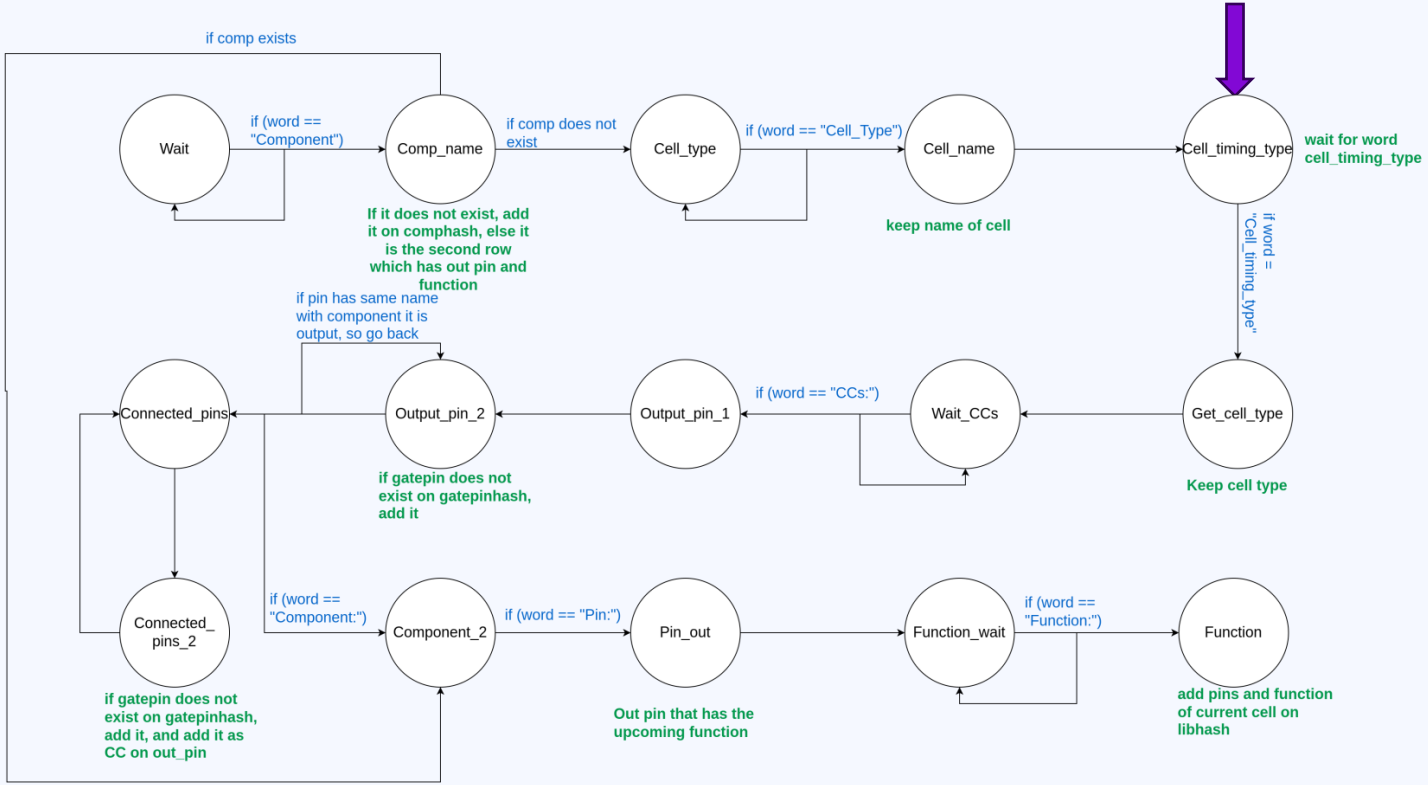
add2d1

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Cell_Timing_Type



Comp_name

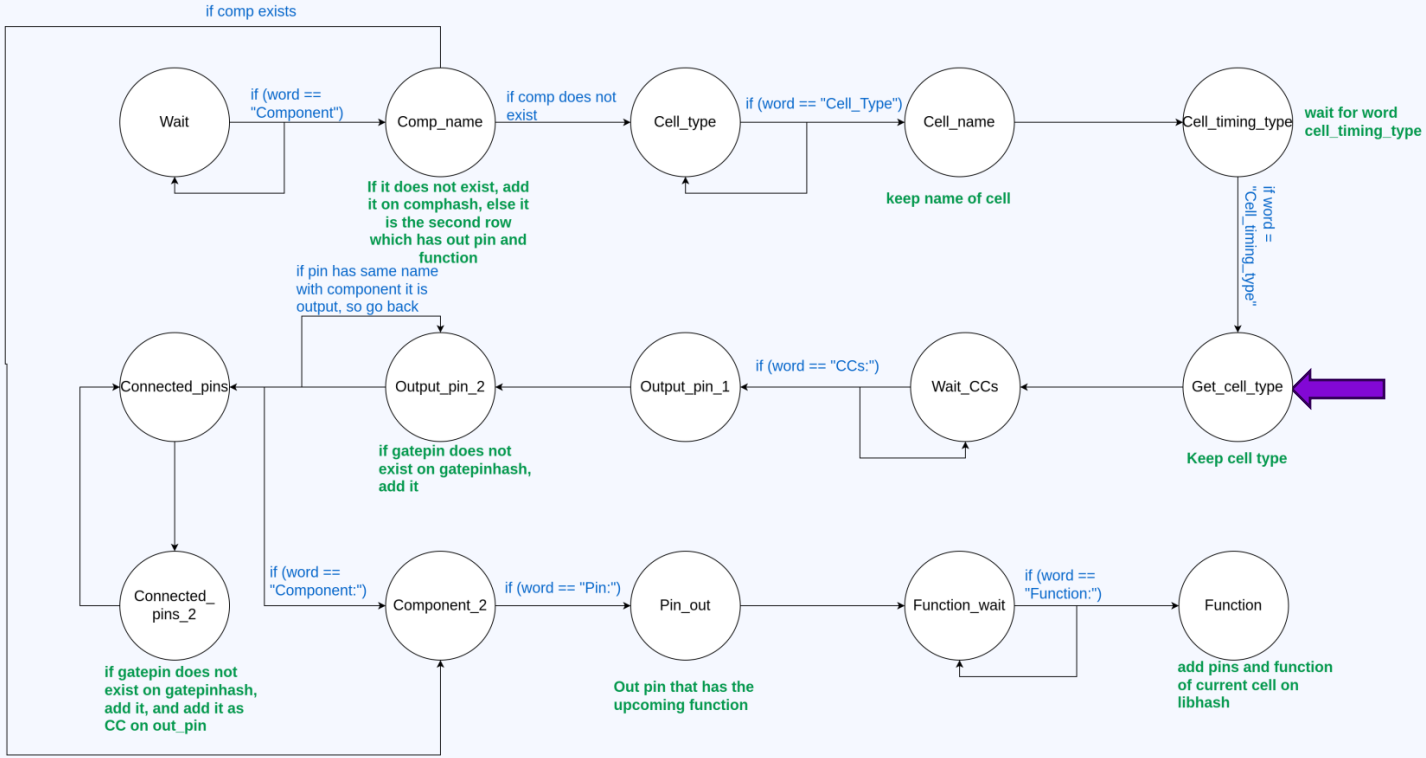
add2d1

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Combinational



Comp_name

add2d1

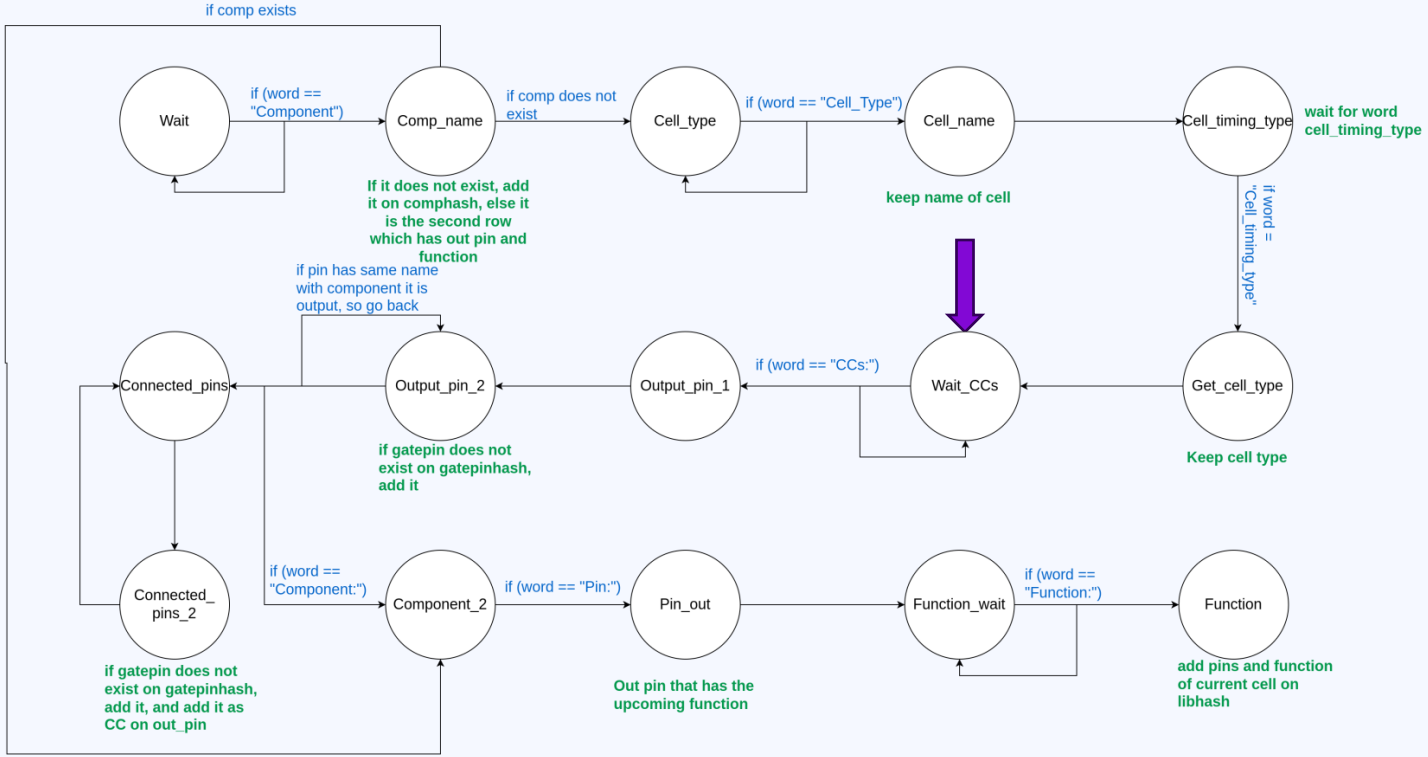
Combinational

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Width



Comp_name

add2d1

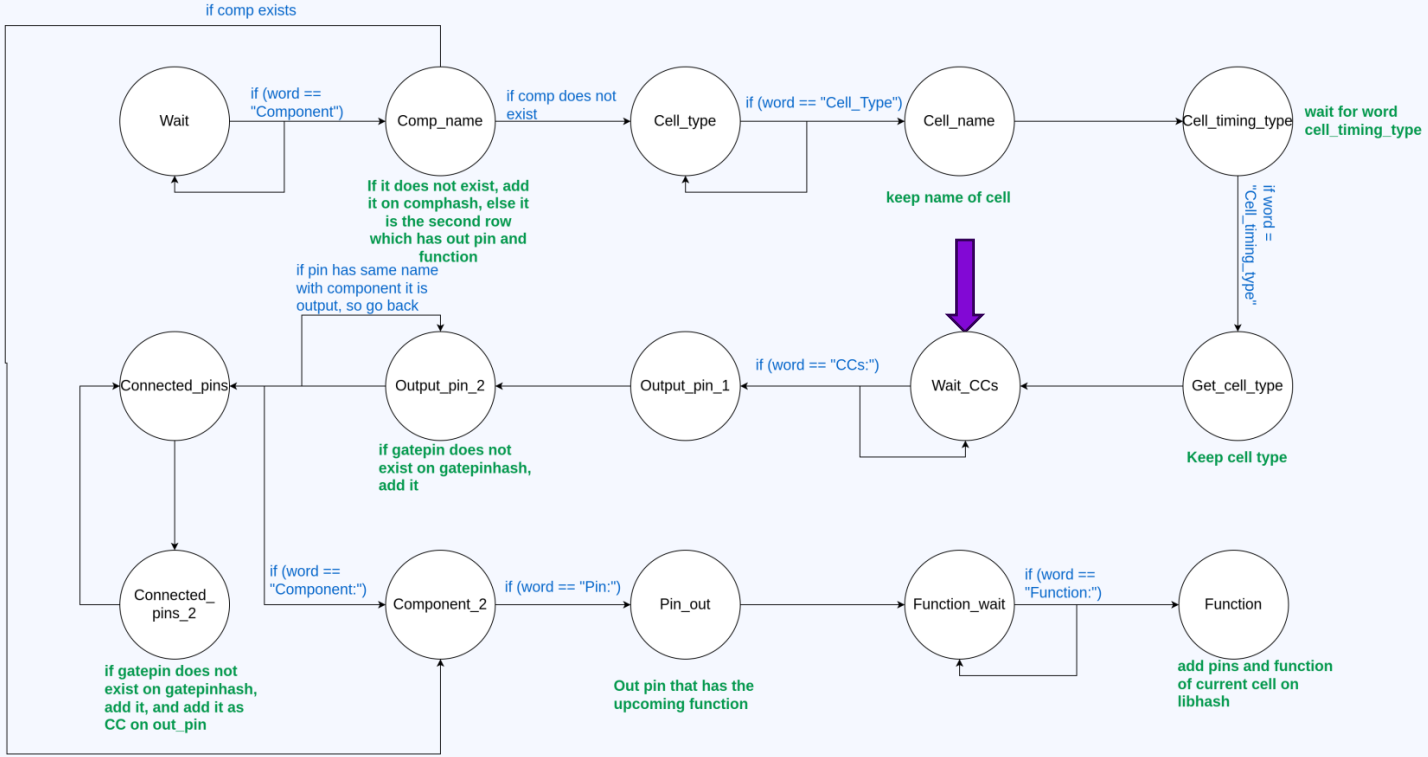
Combinational

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: CCs



Comp_name

add2d1

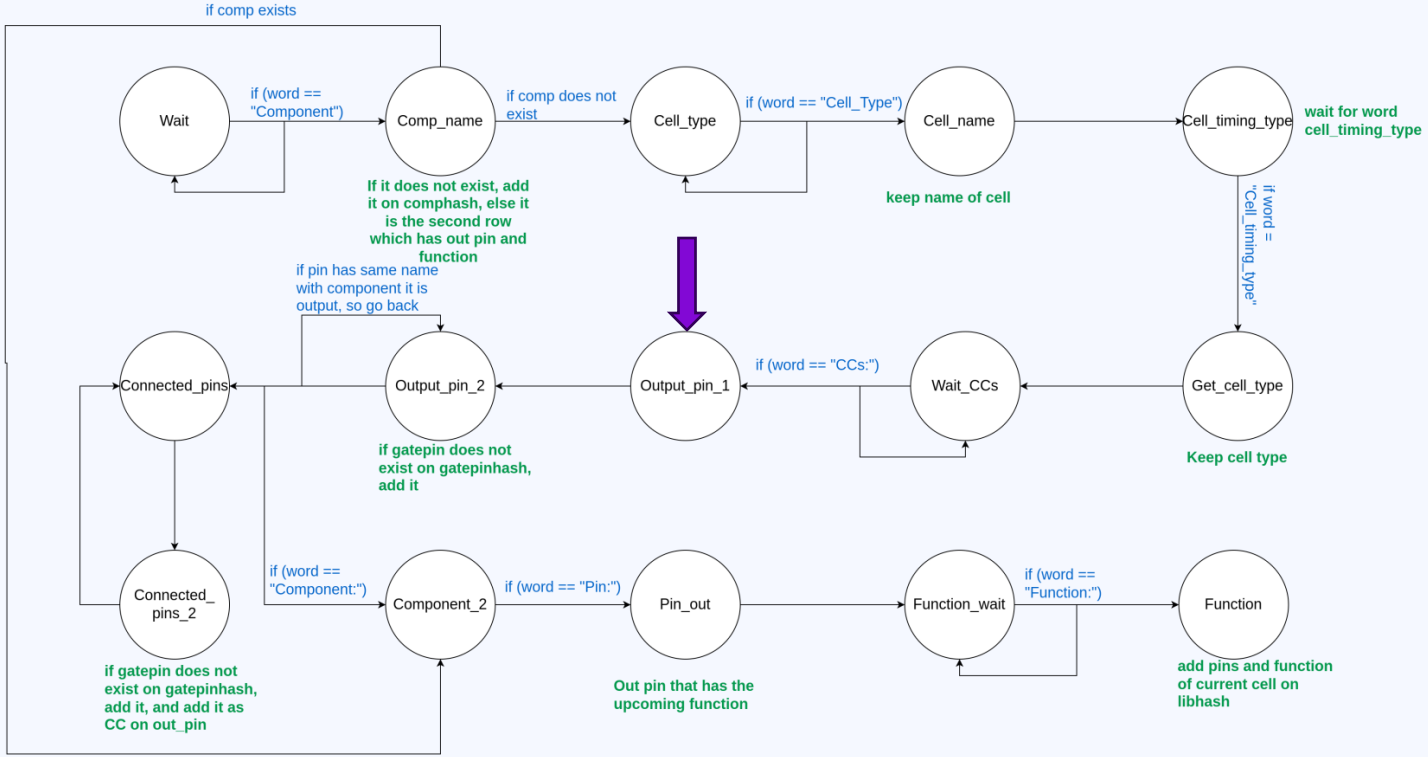
Combinational

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: three_counters/U29



Comp_name

add2d1

Combinational

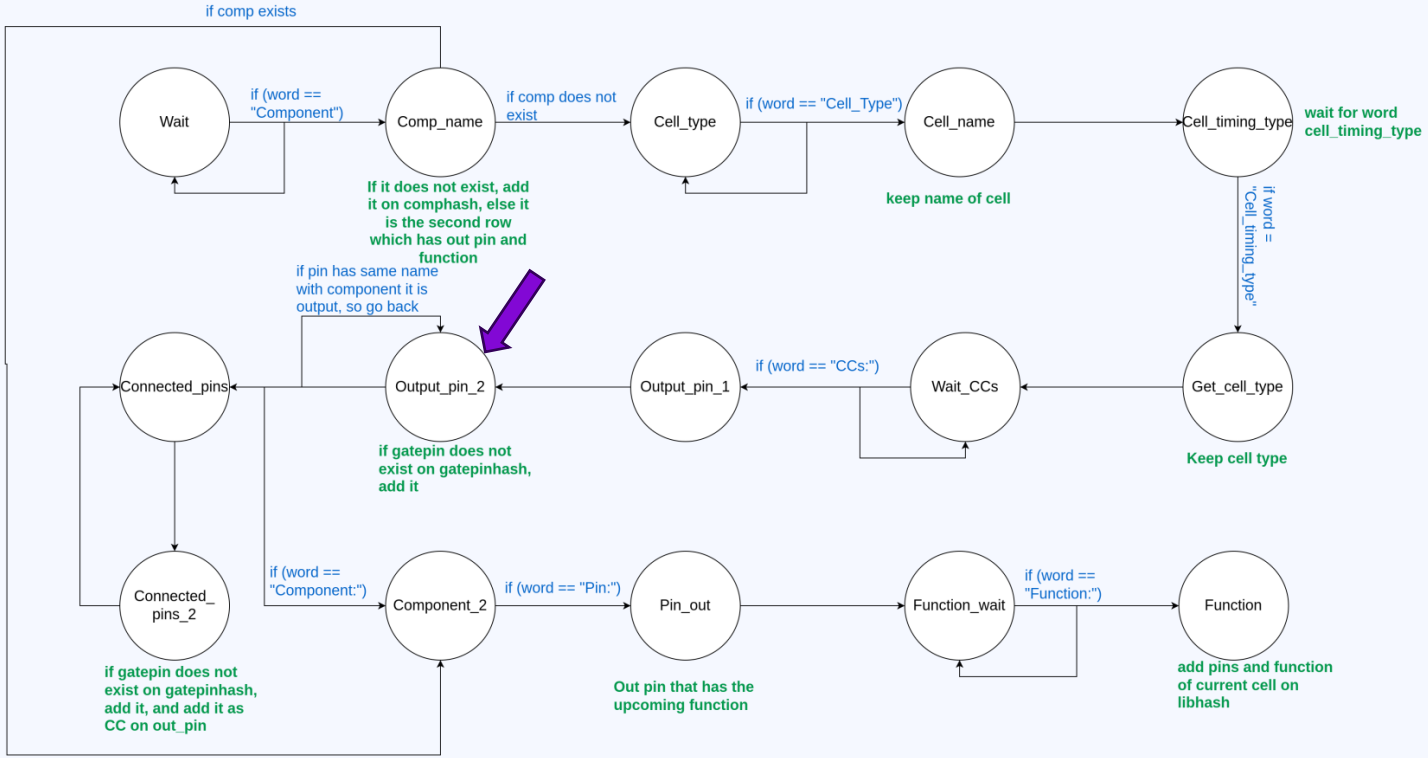
Three_counters/U29

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: (/CO)



Comp_name

add2d1

Combinational

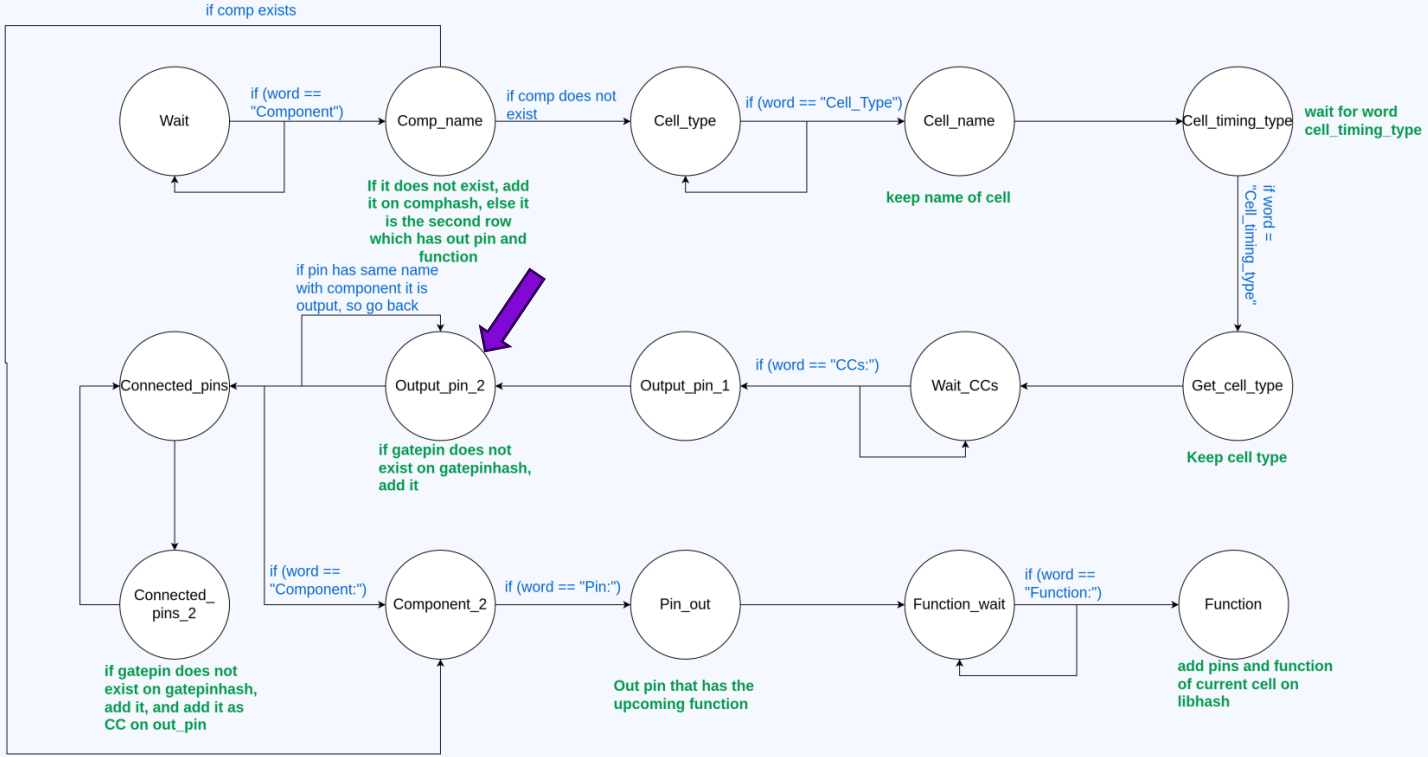
Three_counters/U29/CO

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: (/CO)



Comp_name

add2d1

Combinational

Three_counters /U29/CO

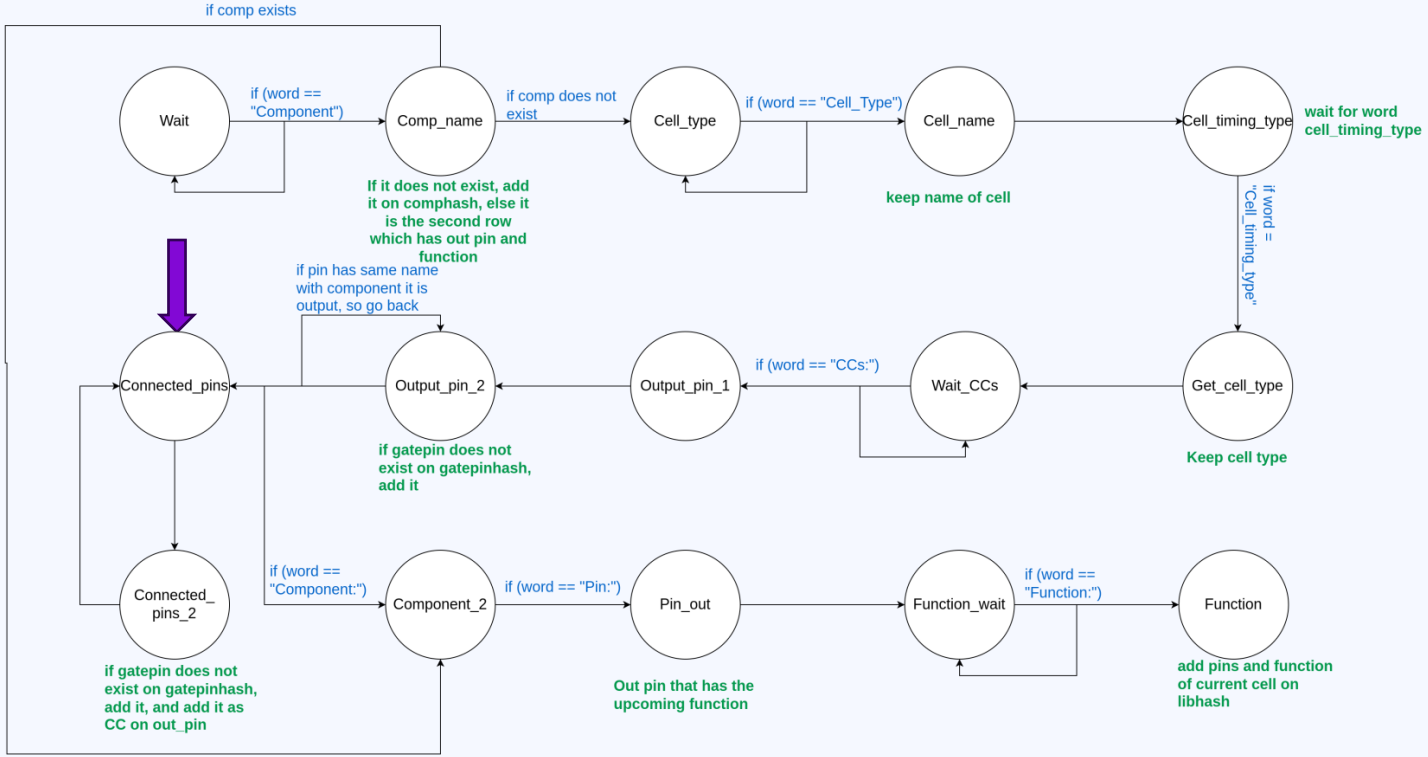
Add gatepin to gatepinhash if it does not exists

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: three_counters/U30



Comp_name

add2d1

Combinational

Three_counters /U29/CO

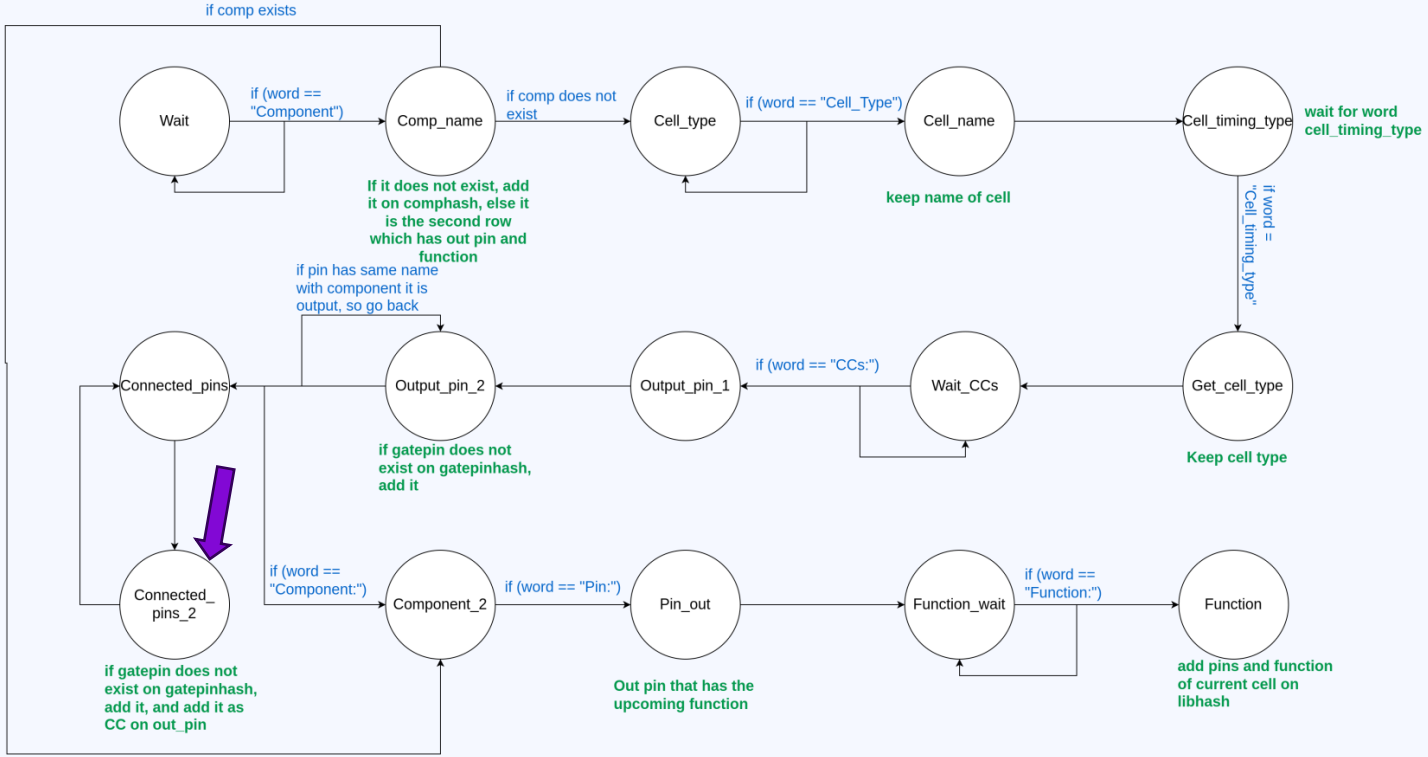
Three_counters /U30

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: (/CI)



Comp_name

add2d1

Combinational

Three_counters /U29/CO

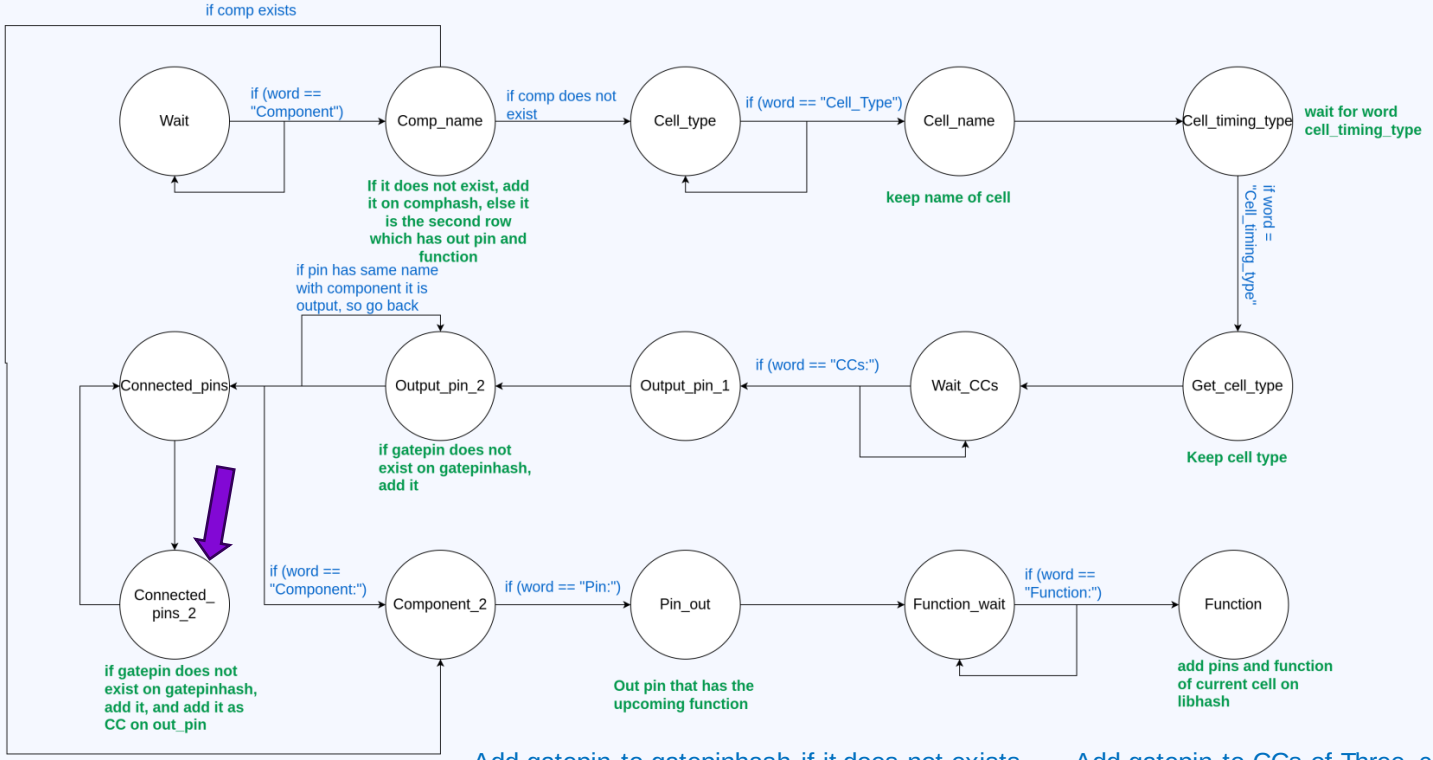
Three_counters /U30/CI

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: (/CI)



Comp_name

add2d1

Combinational

✓ Three_counters /U29/CO

✓ Three_counters /U30/CI

Add gatepin to gatepinhash if it does not exists

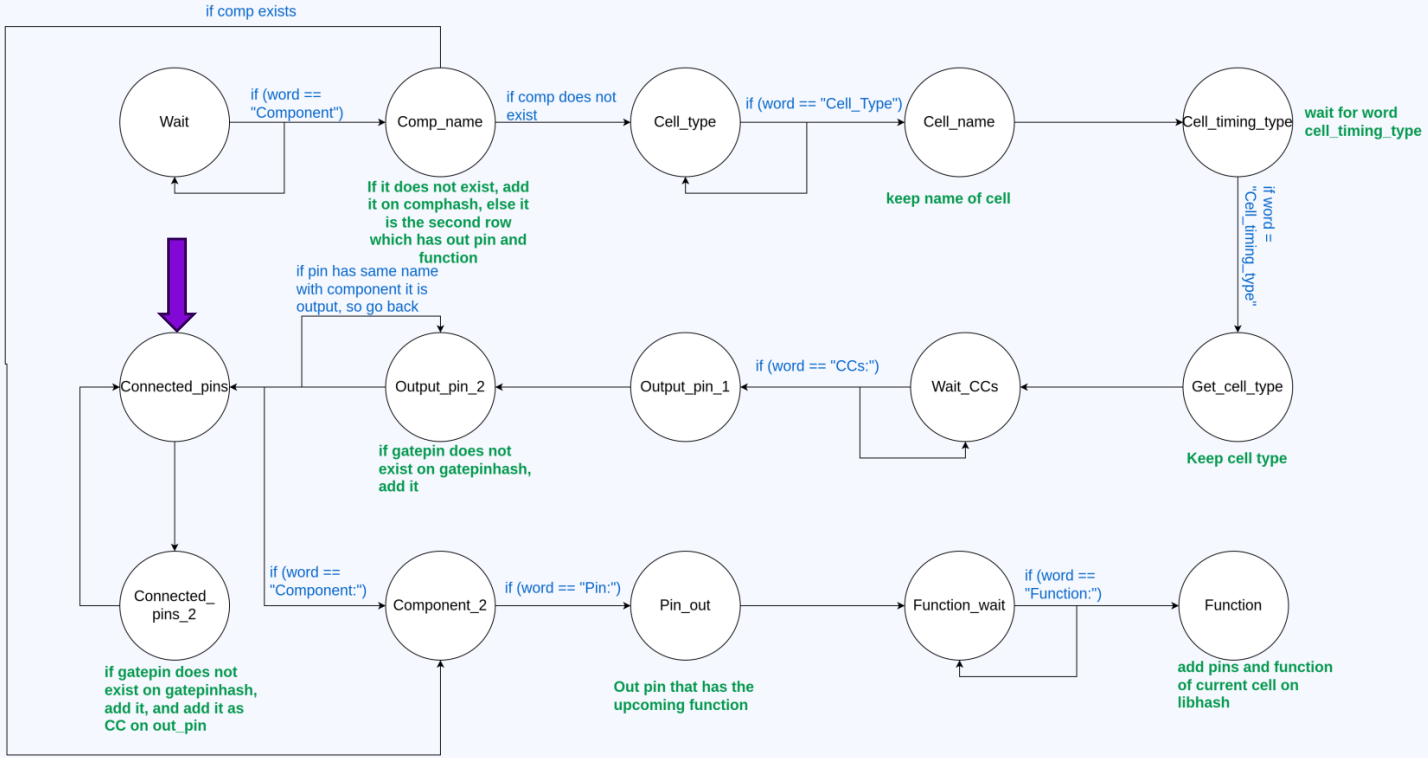
Add gatepin to CCs of Three_counters/U29/CO

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: three_counters/U29



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

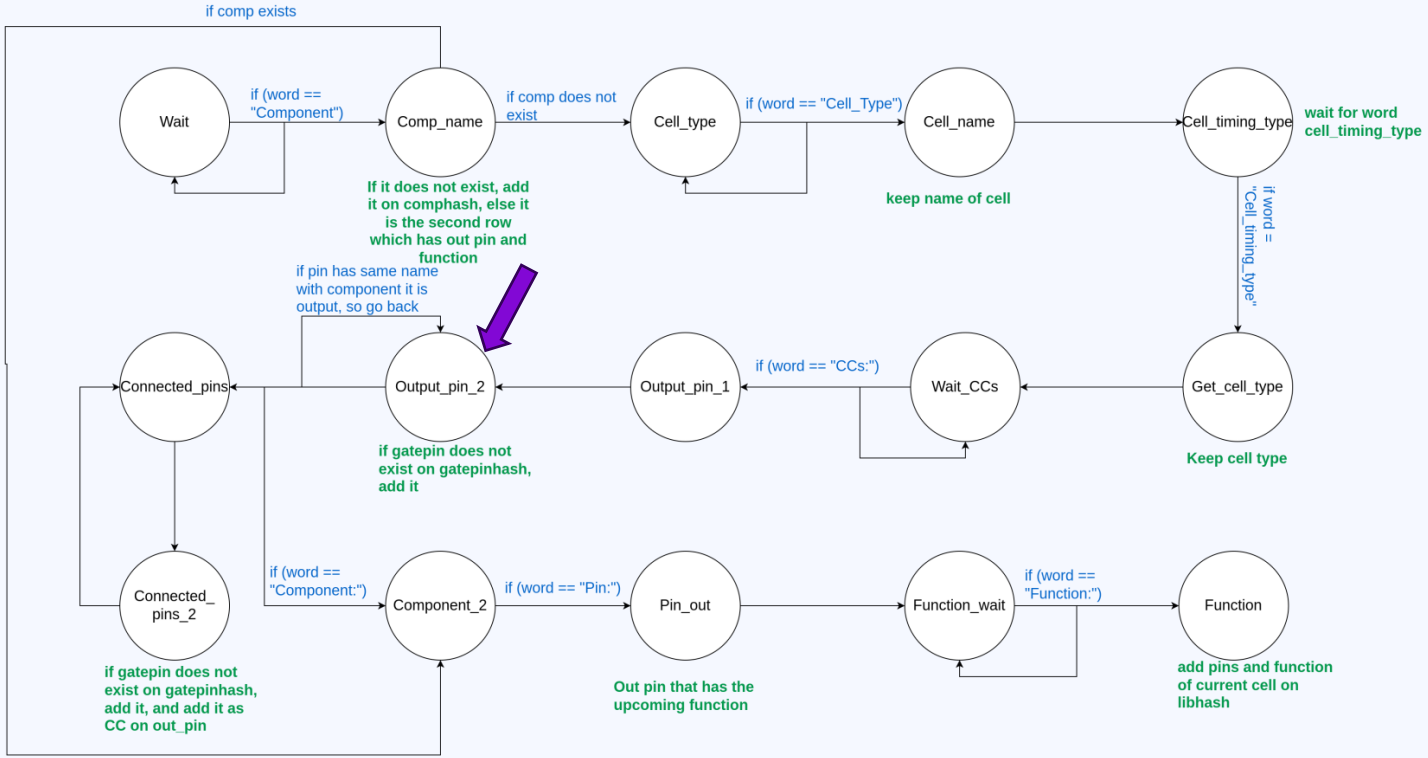
Three_counters /U29

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: (/SM)



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

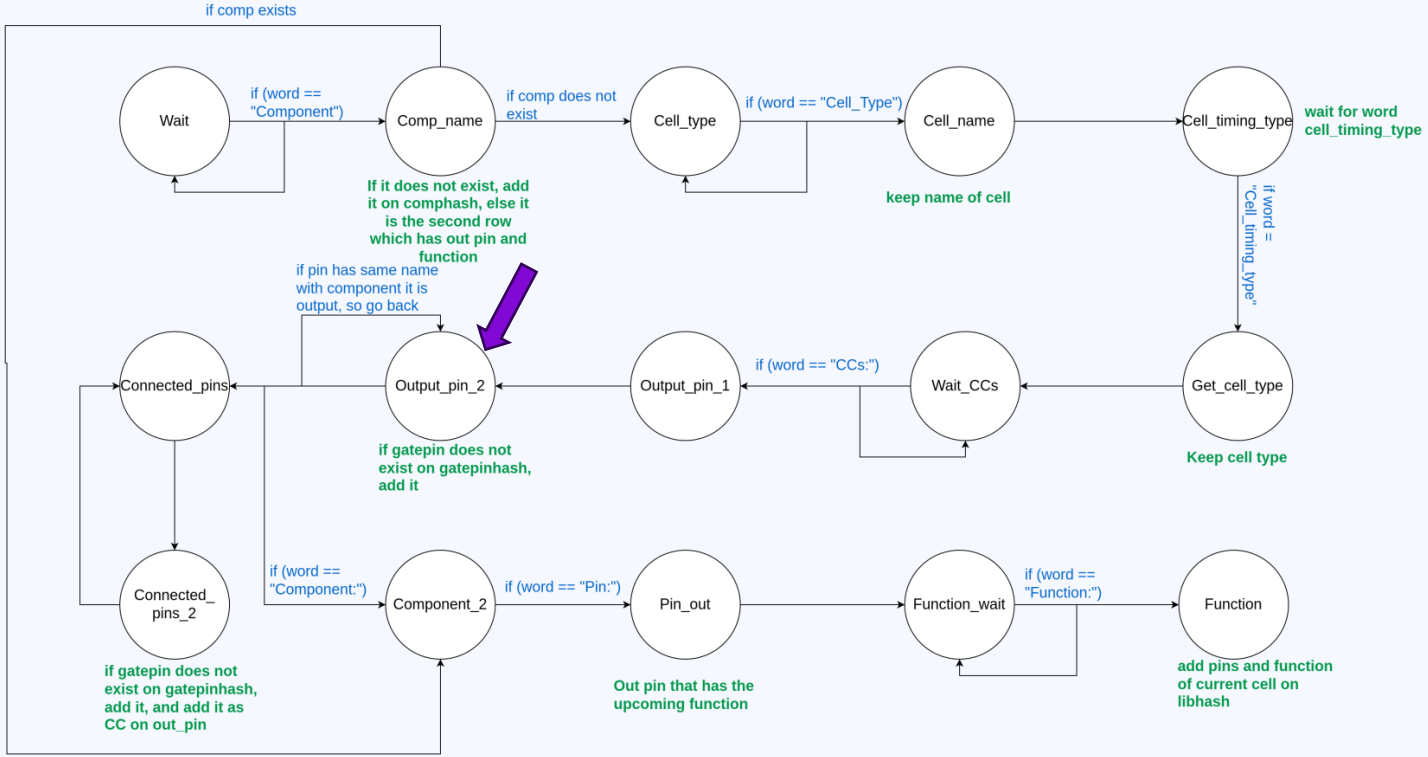
Three_counters /U29/SM

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: (/SM)



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

✓ Three_counters /U29/SM

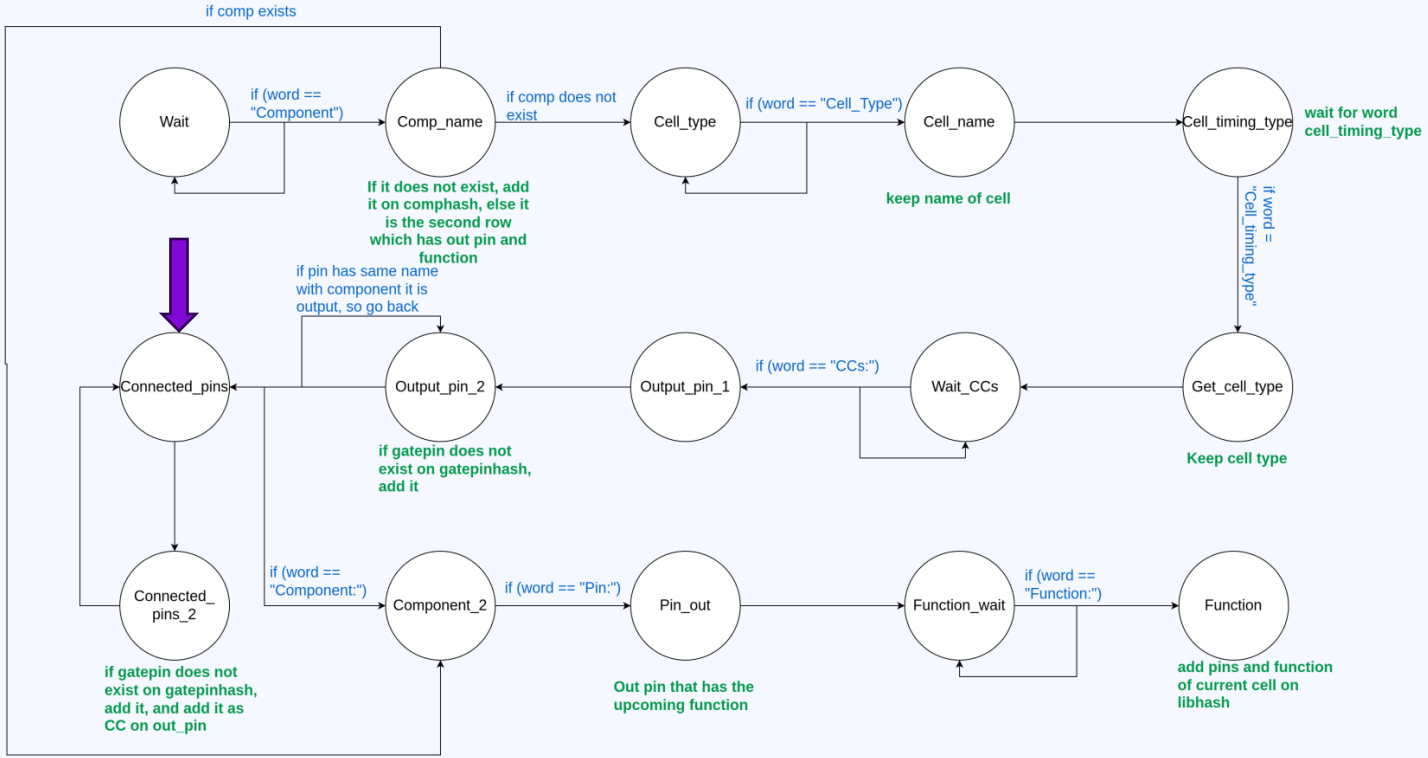
Add three_counters/U29/SM to gatepinhash if it does not exists

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: three_counters/U116



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

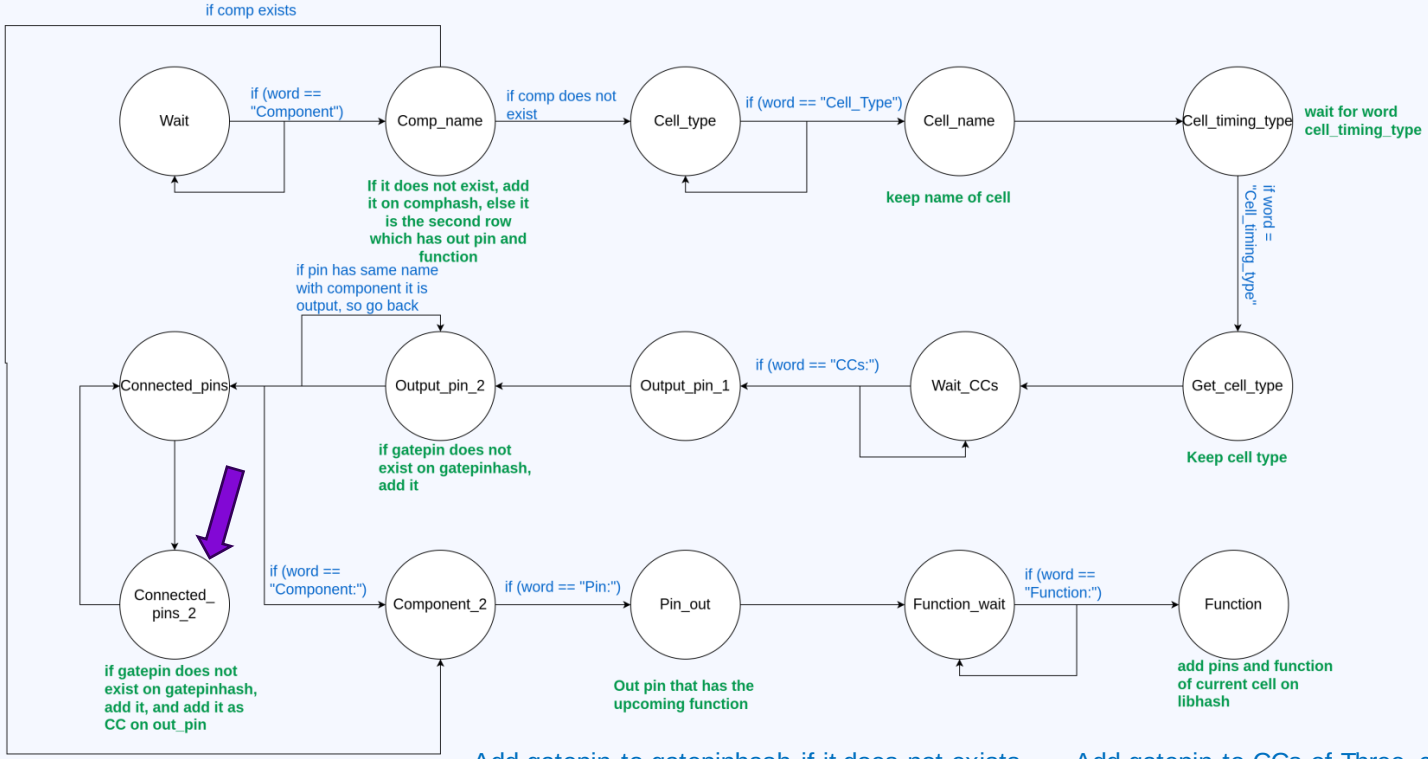
Three_counters /U116

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: (/I1)



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

✓ Three_counters /U116/I1

Add gatepin to gatepinhash if it does not exists

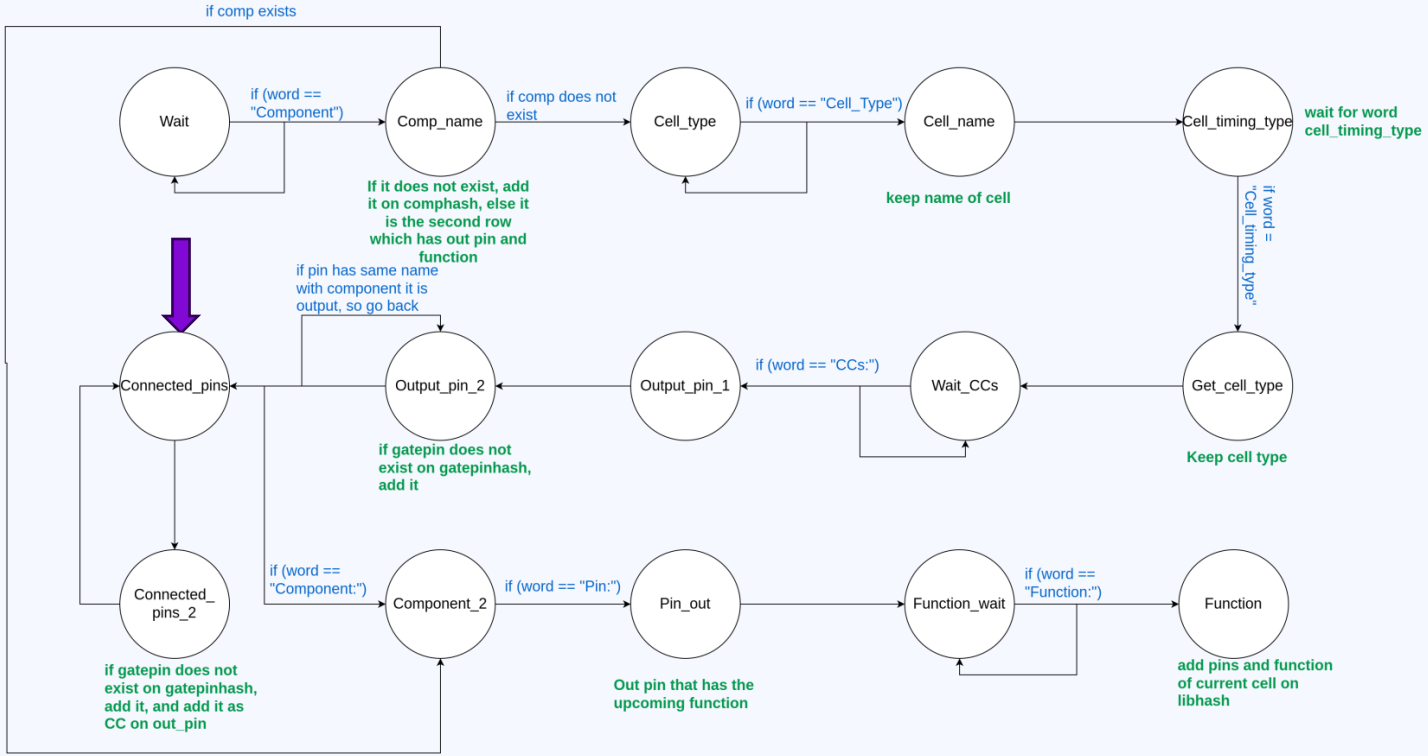
Add gatepin to CCs of Three_counters/U29/SM

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Component



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

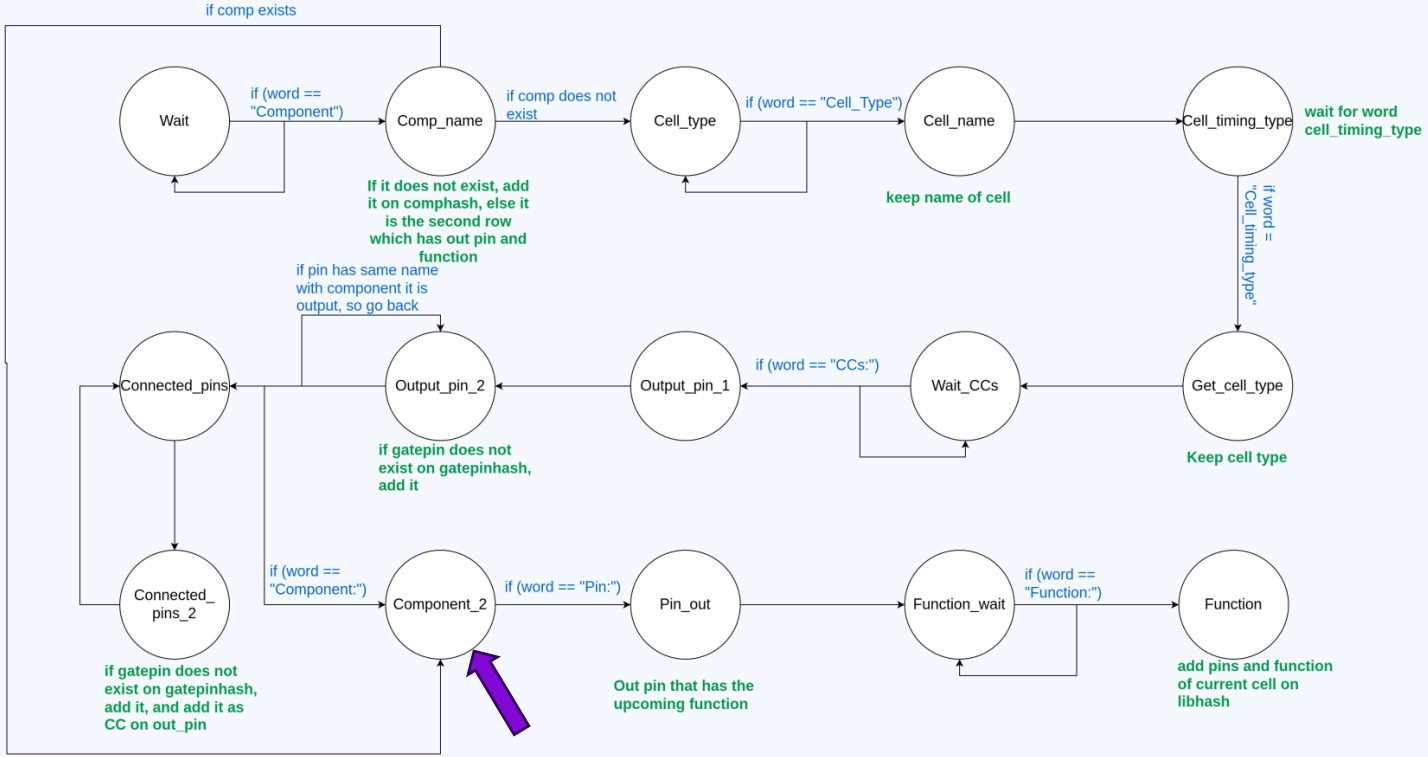
Three_counters /U116/I1

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: three_counters/U29,



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

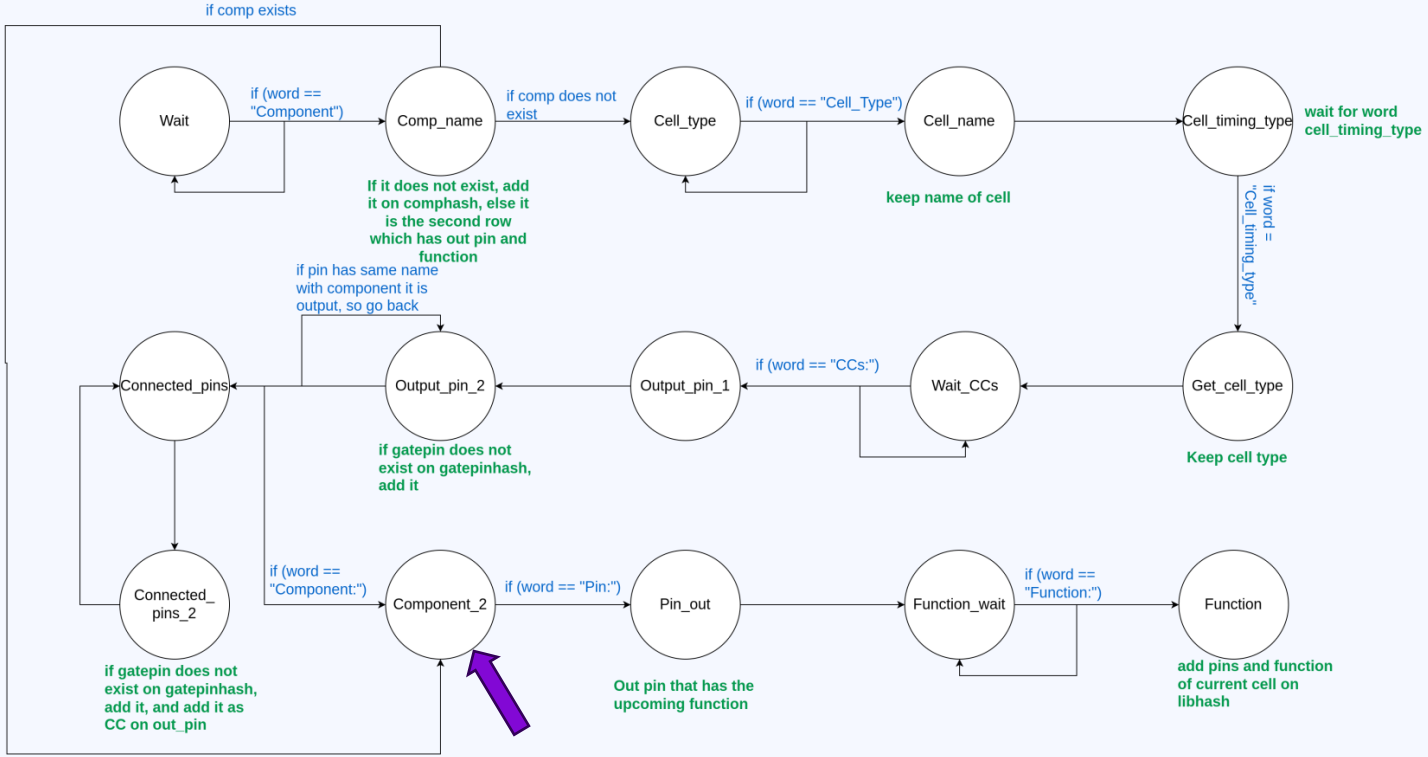
Three_counters /U116/I1

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Output



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

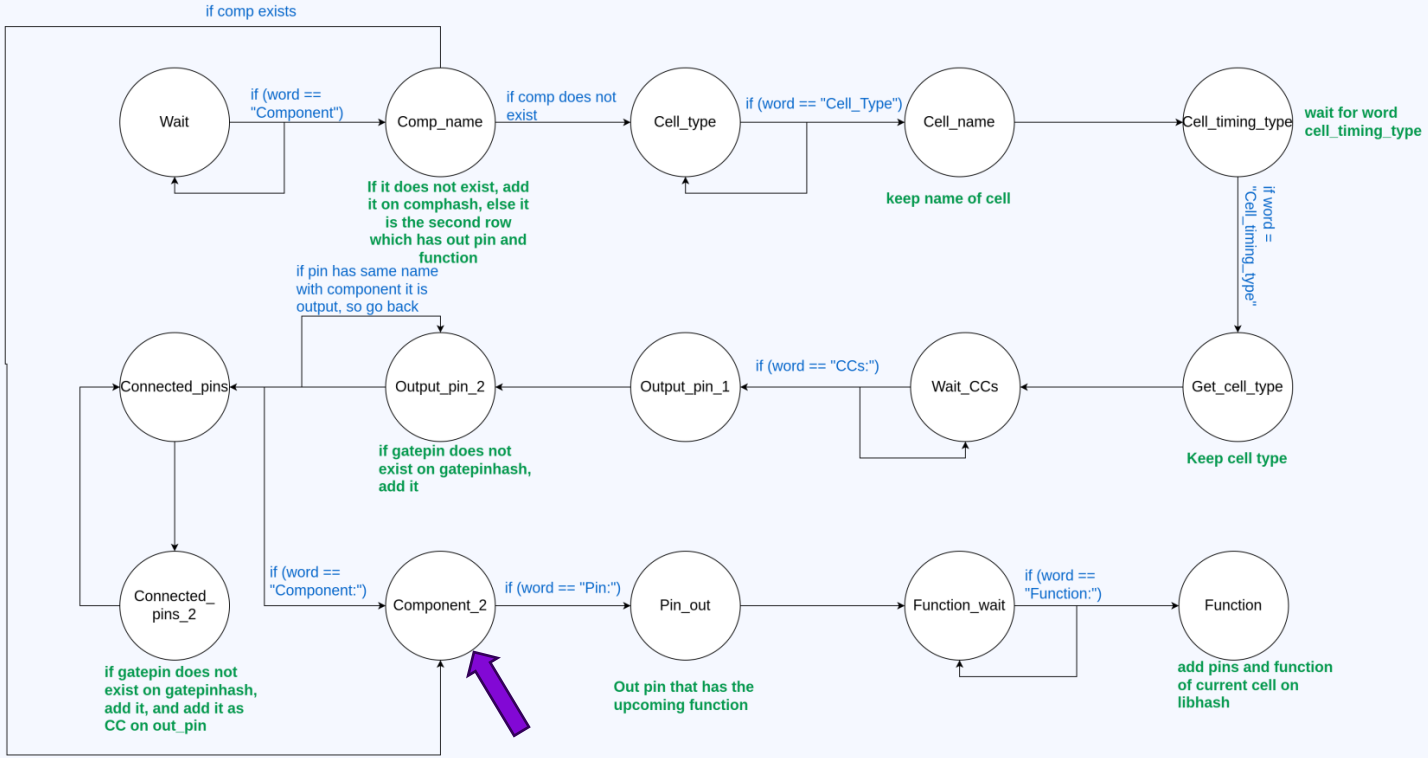
Three_counters /U116/I1

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Pin:



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

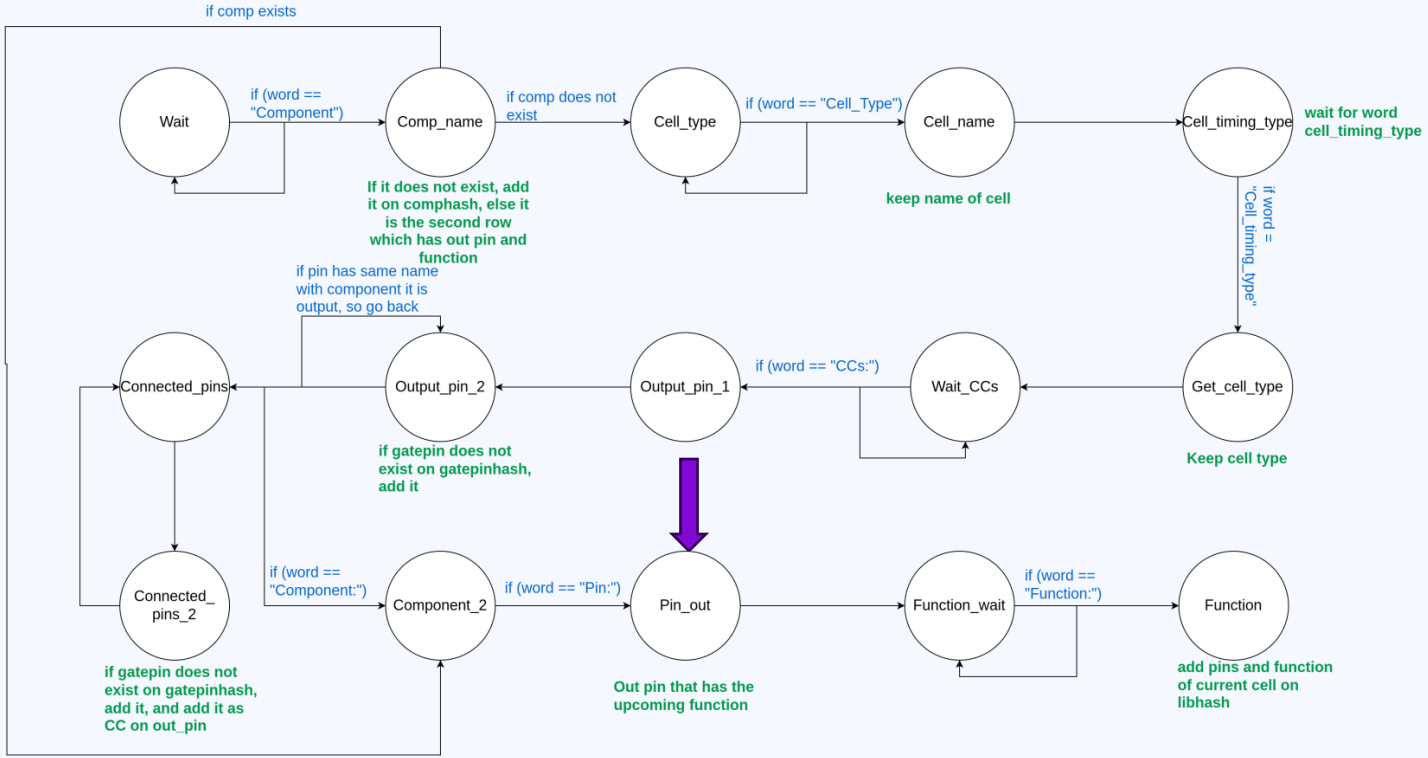
Three_counters /U116/I1

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: CO,



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

Three_counters /U116/I1

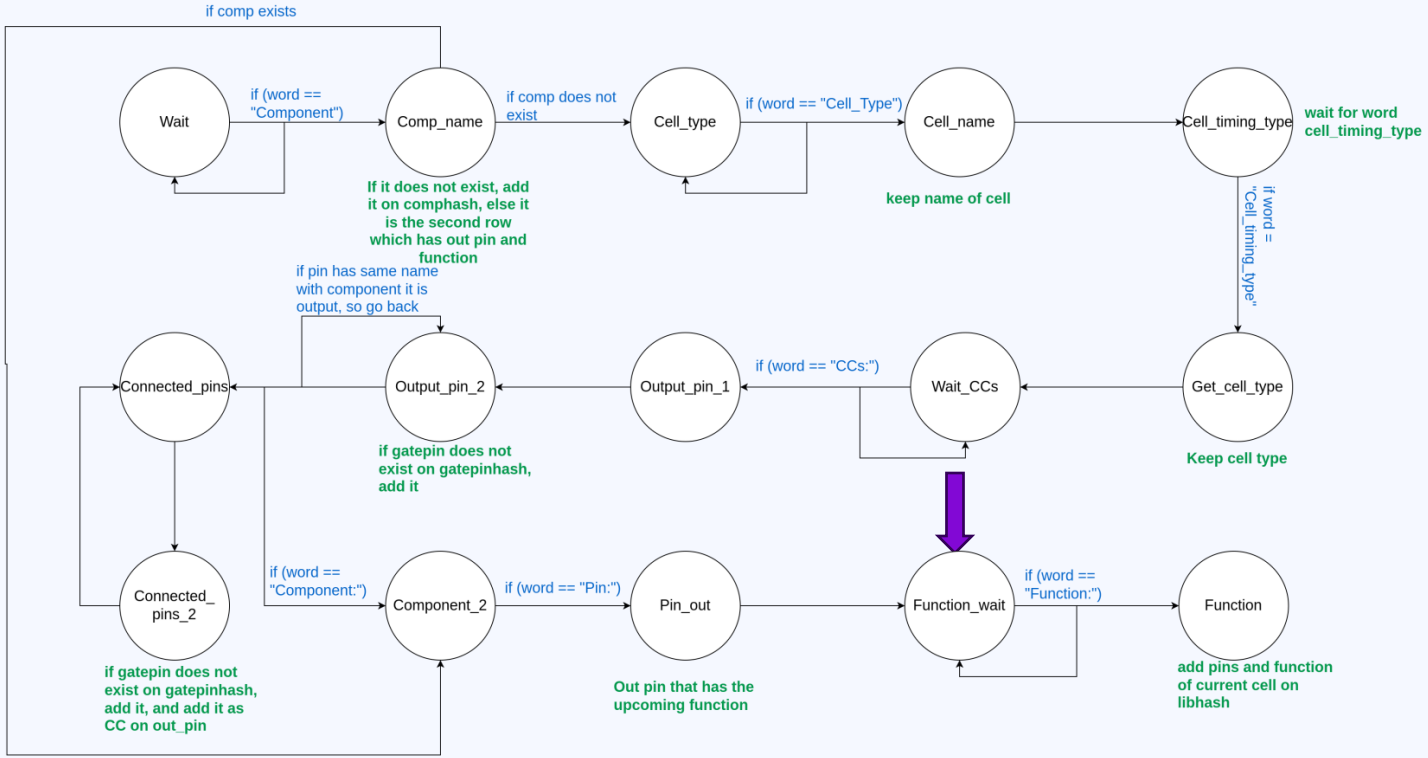
Cell_pin

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Boolean



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

Three_counters /U116/I1

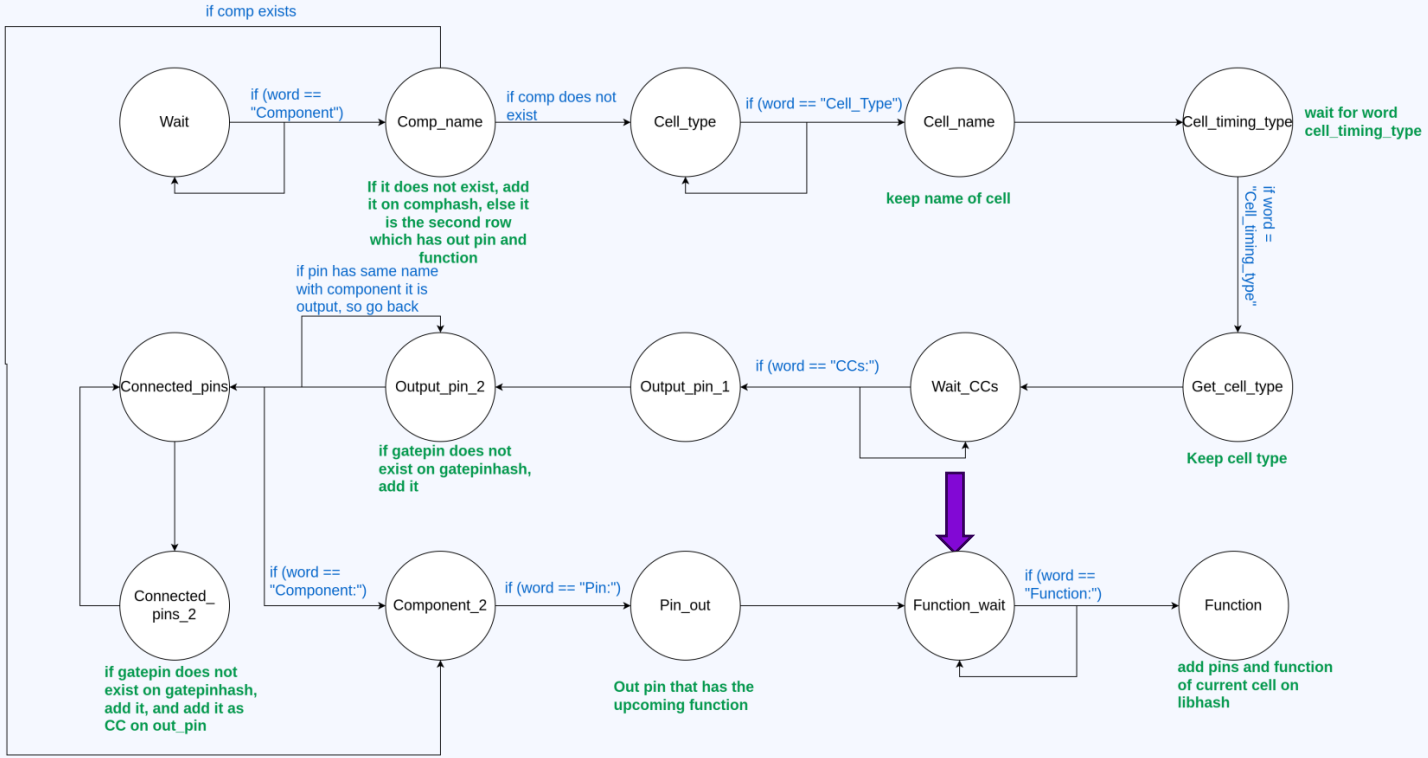
Cell_pin

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Function



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

Three_counters /U116/I1

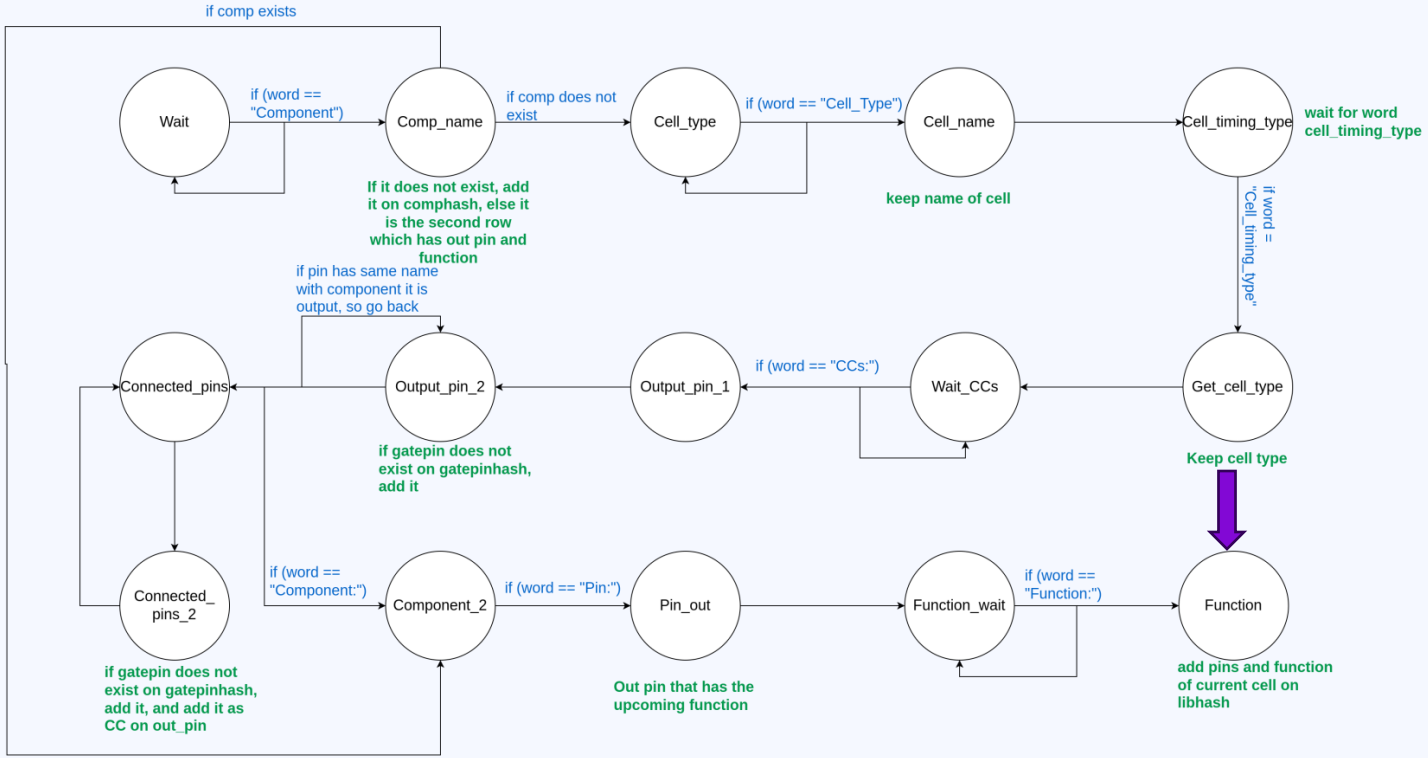
Cell_pin

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: (A*B)+(A*CI)+(B*CI)



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

Three_counters /U116/I1

Cell_pin

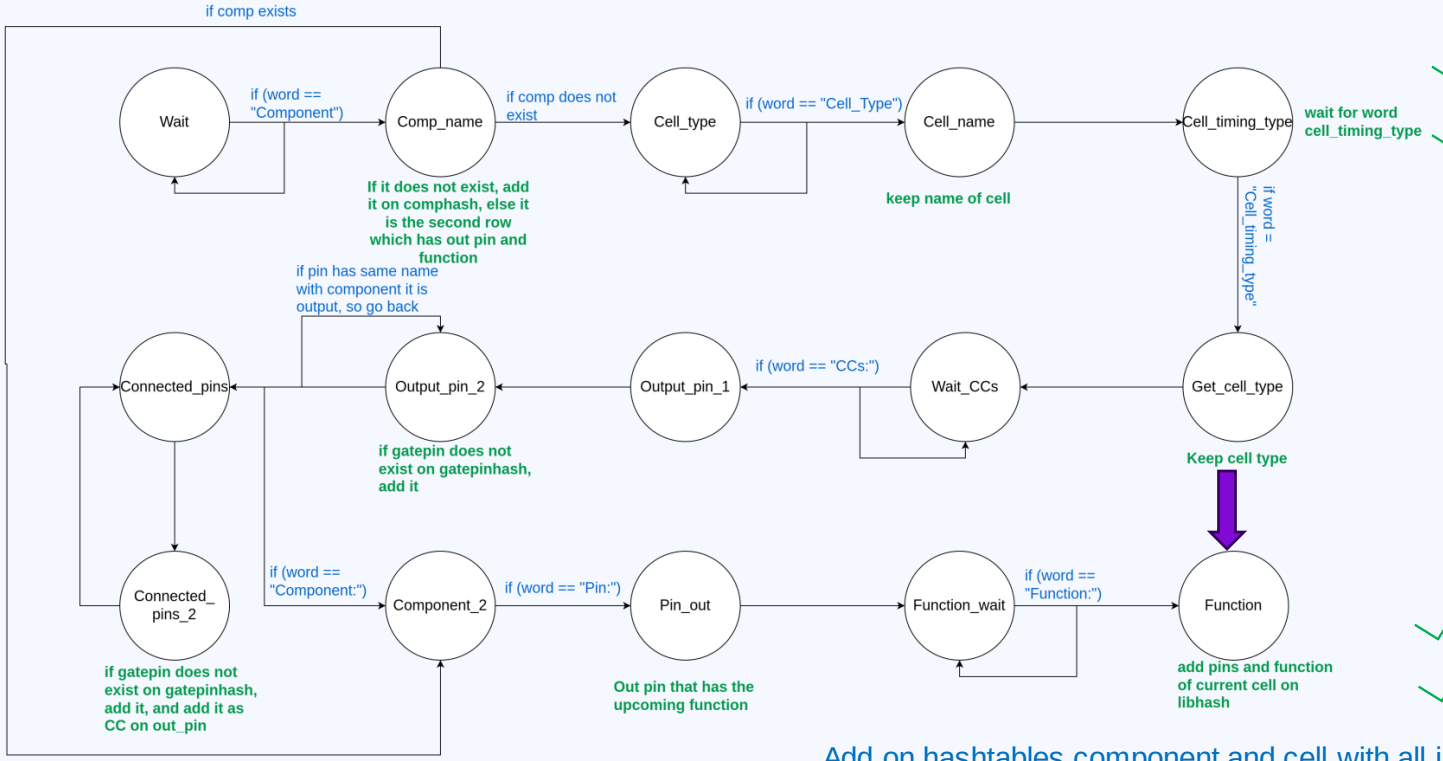
(A*B)+(A*CI)+(B*CI)

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: (A*B)+(A*CI)+(B*CI)



Comp_name

✓ add2d1

✓ Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

Three_counters /U116/I1

✓ Cell_pin

✓ (A*B)+(A*CI)+(B*CI)

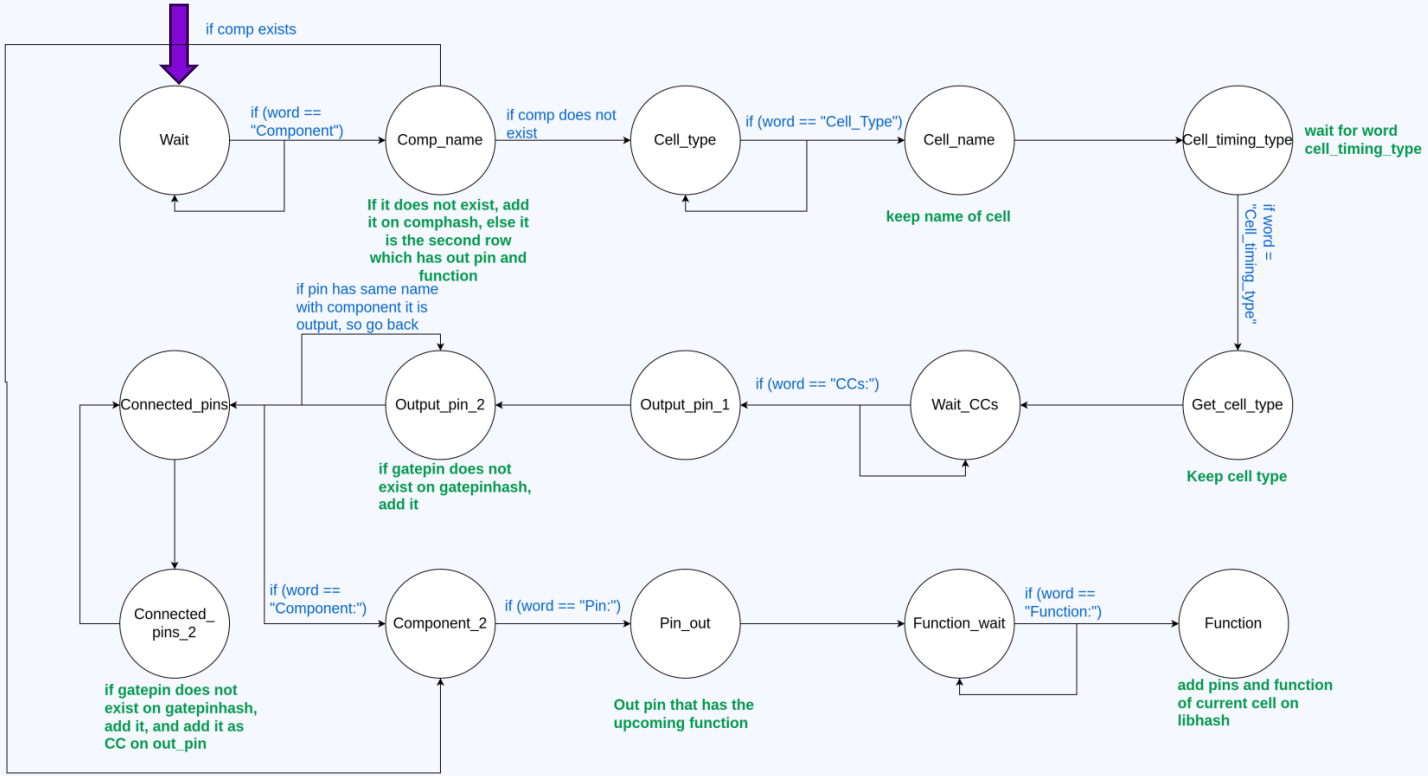
Add on hashtables component and cell with all information

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Component:



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

Three_counters /U116/I1

Cell_pin

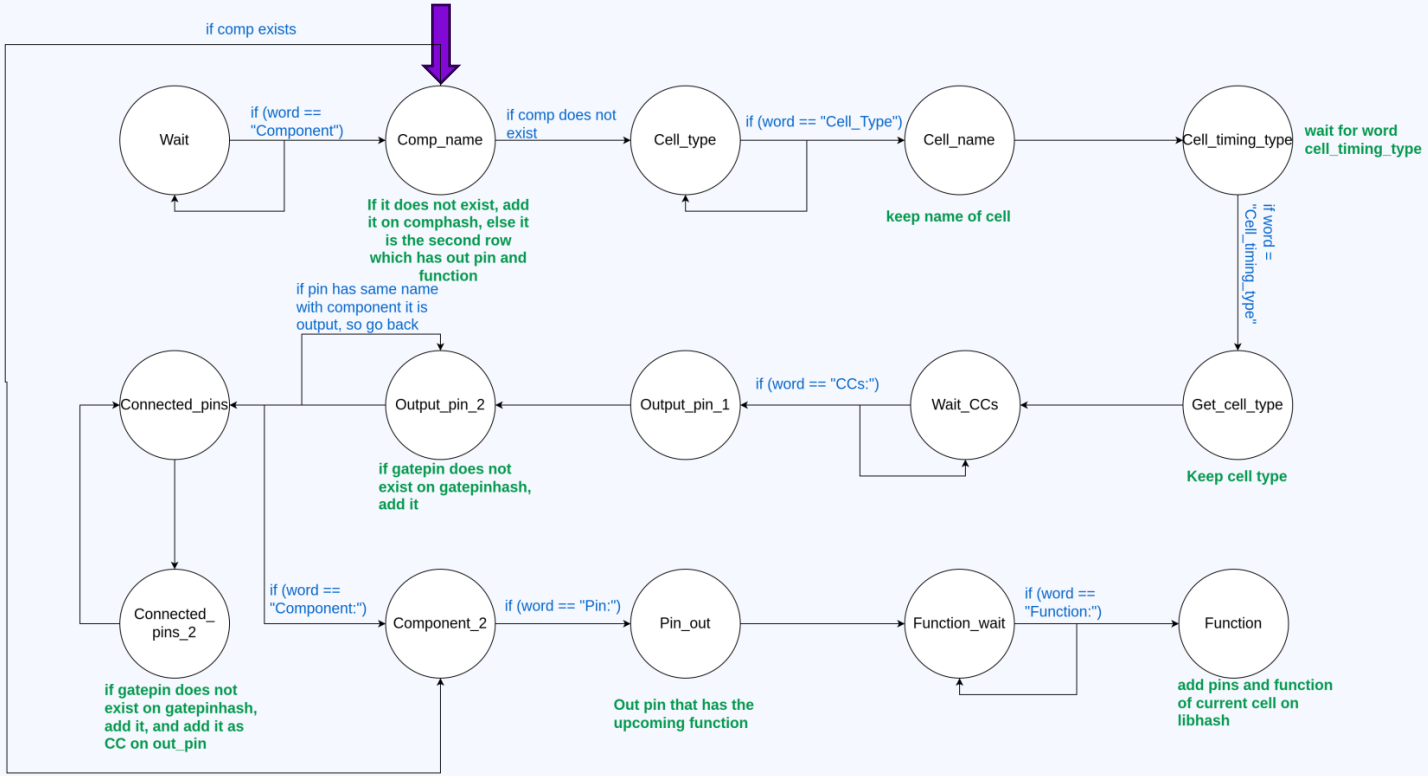
(A*B)+(A*CI)+(B*CI)

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: three_counters/U29, Already exists



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

Three_counters /U116/I1

Cell_pin

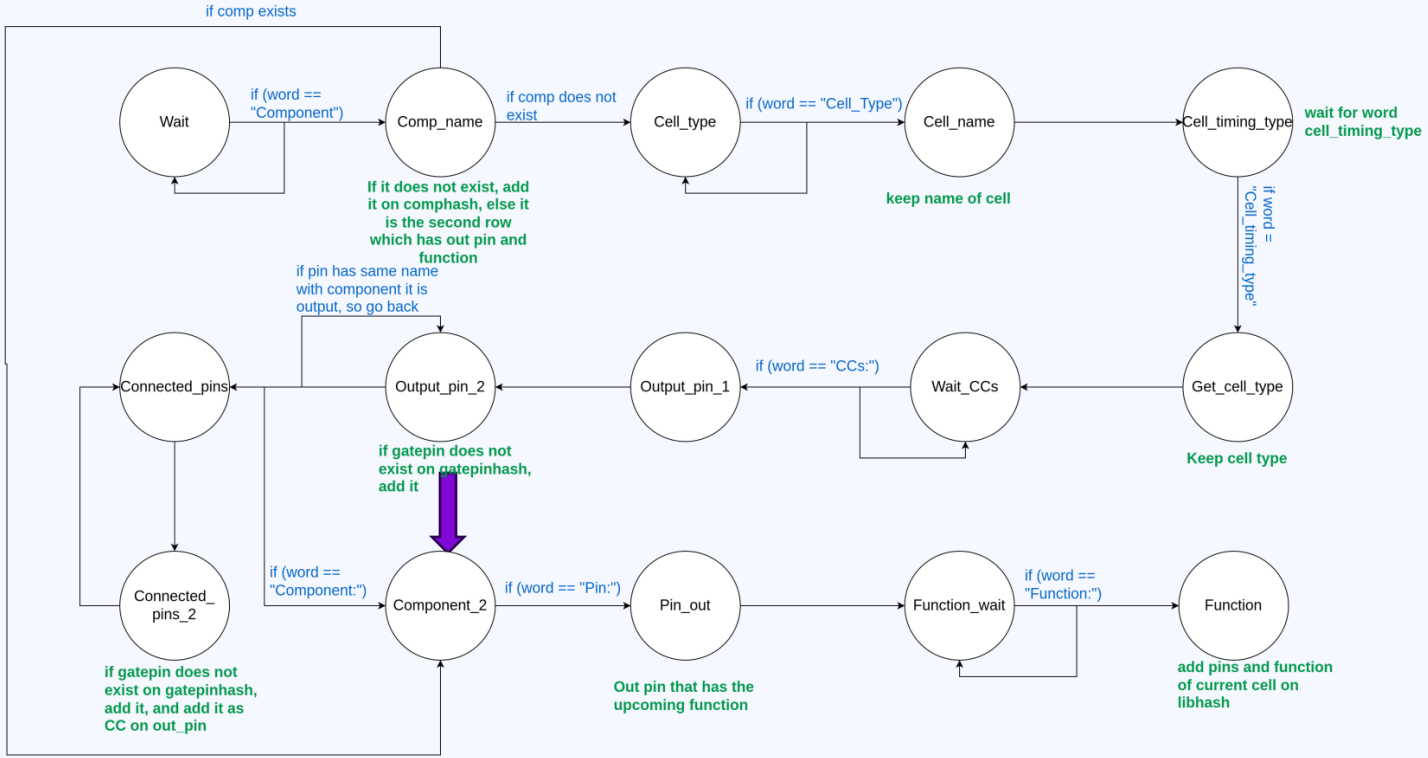
(A*B)+(A*CI)+(B*CI)

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Output



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

Three_counters /U116/I1

Cell_pin

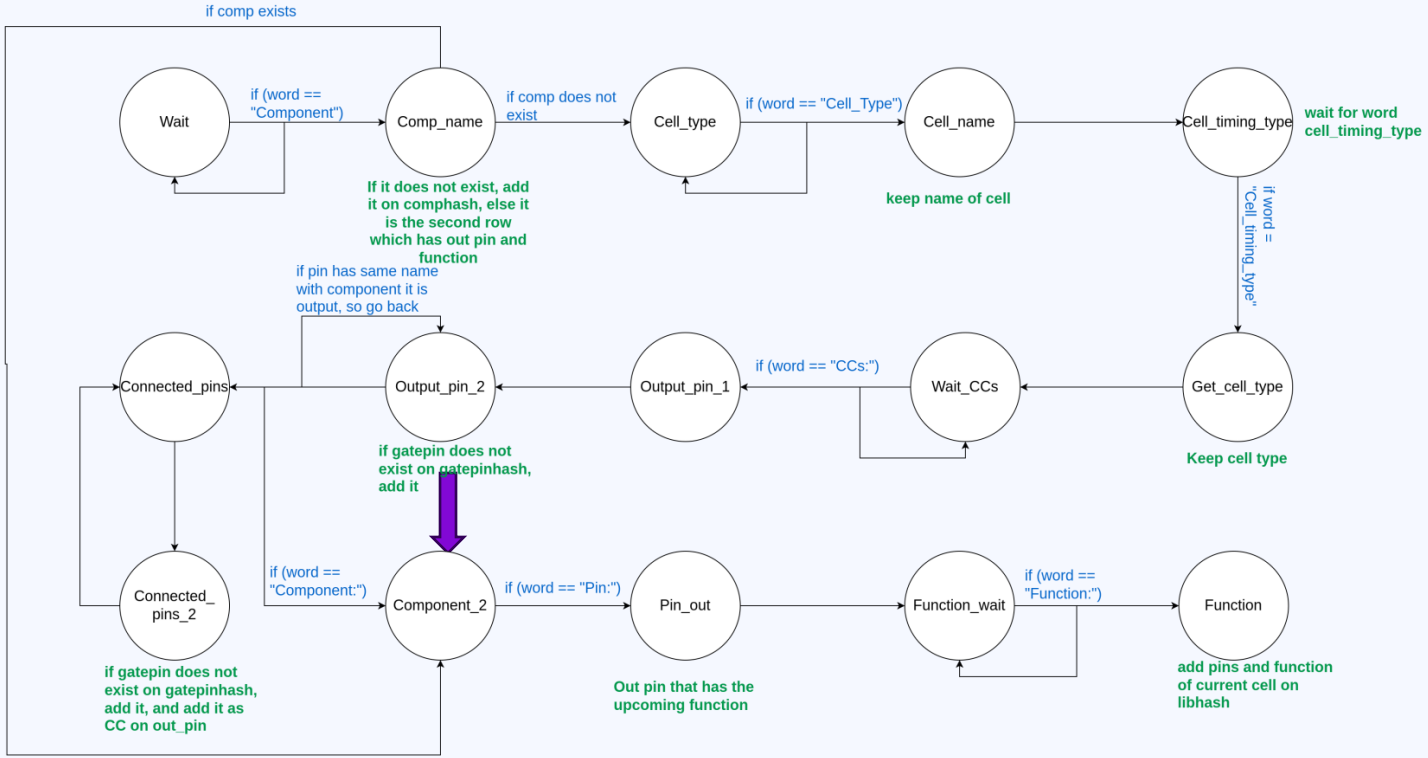
(A*B)+(A*CI)+(B*CI)

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Pin:



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

Three_counters /U116/I1

Cell_pin

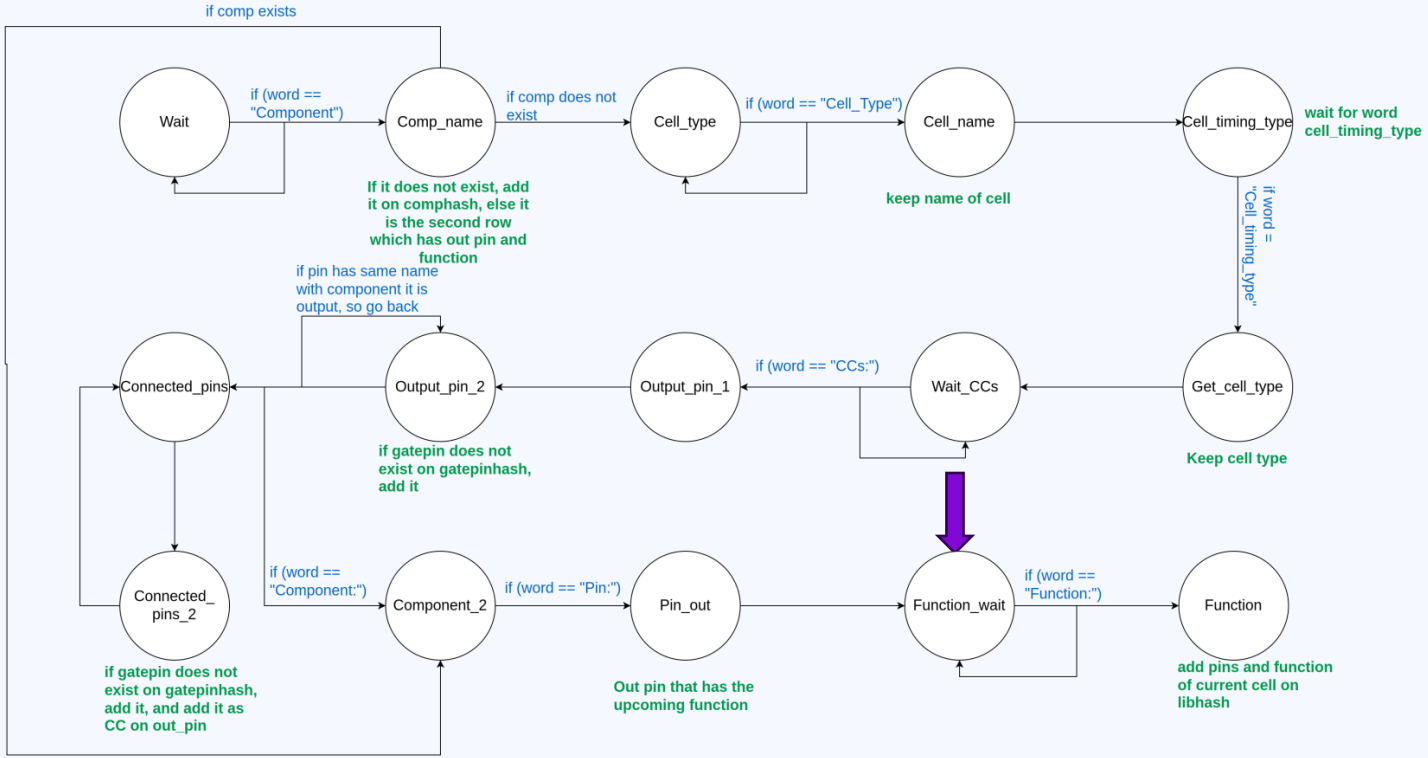
(A*B)+(A*CI)+(B*CI)

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: SM,



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

Three_counters /U116/I1

Cell_pin (SM)

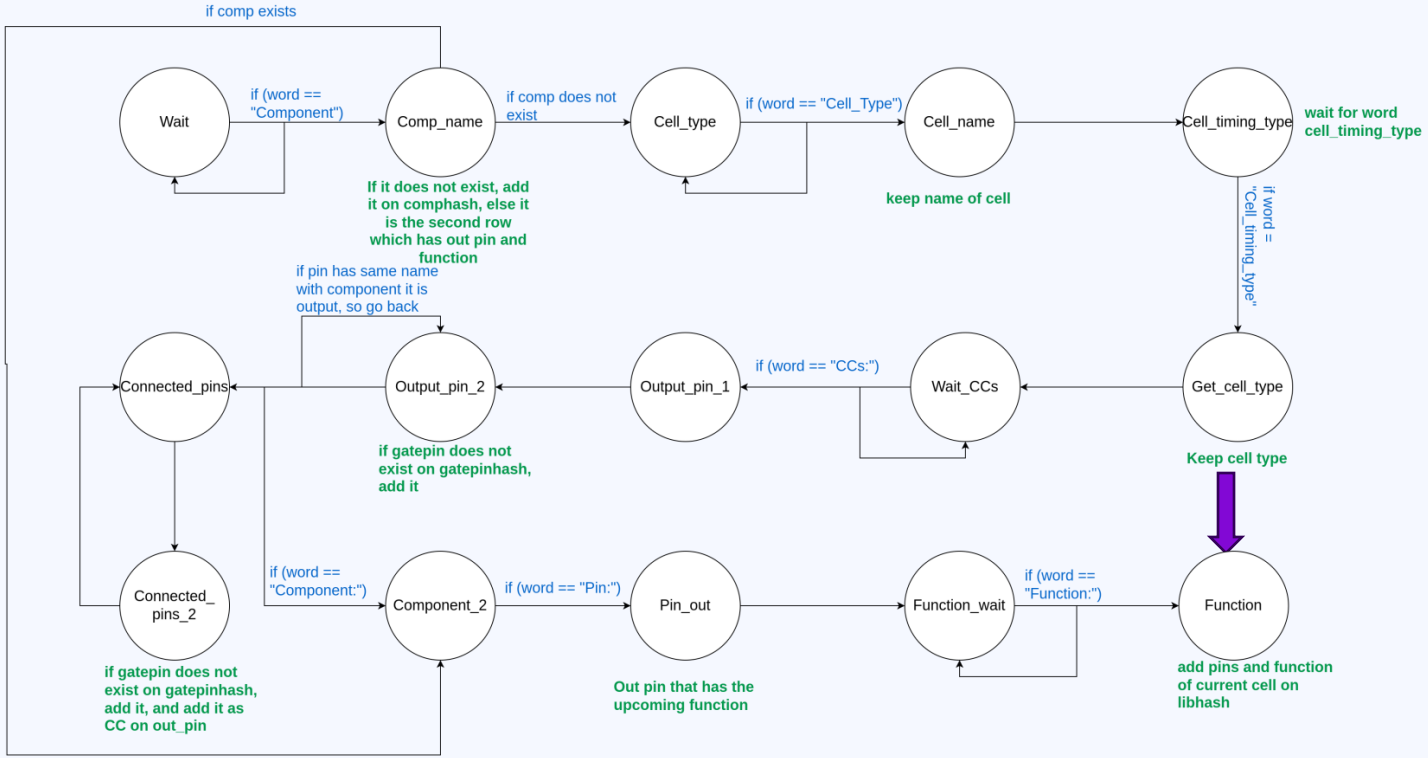
(A*B)+(A*CI)+(B*CI)

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: Function



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

Three_counters /U116/I1

Cell_pin (SM)

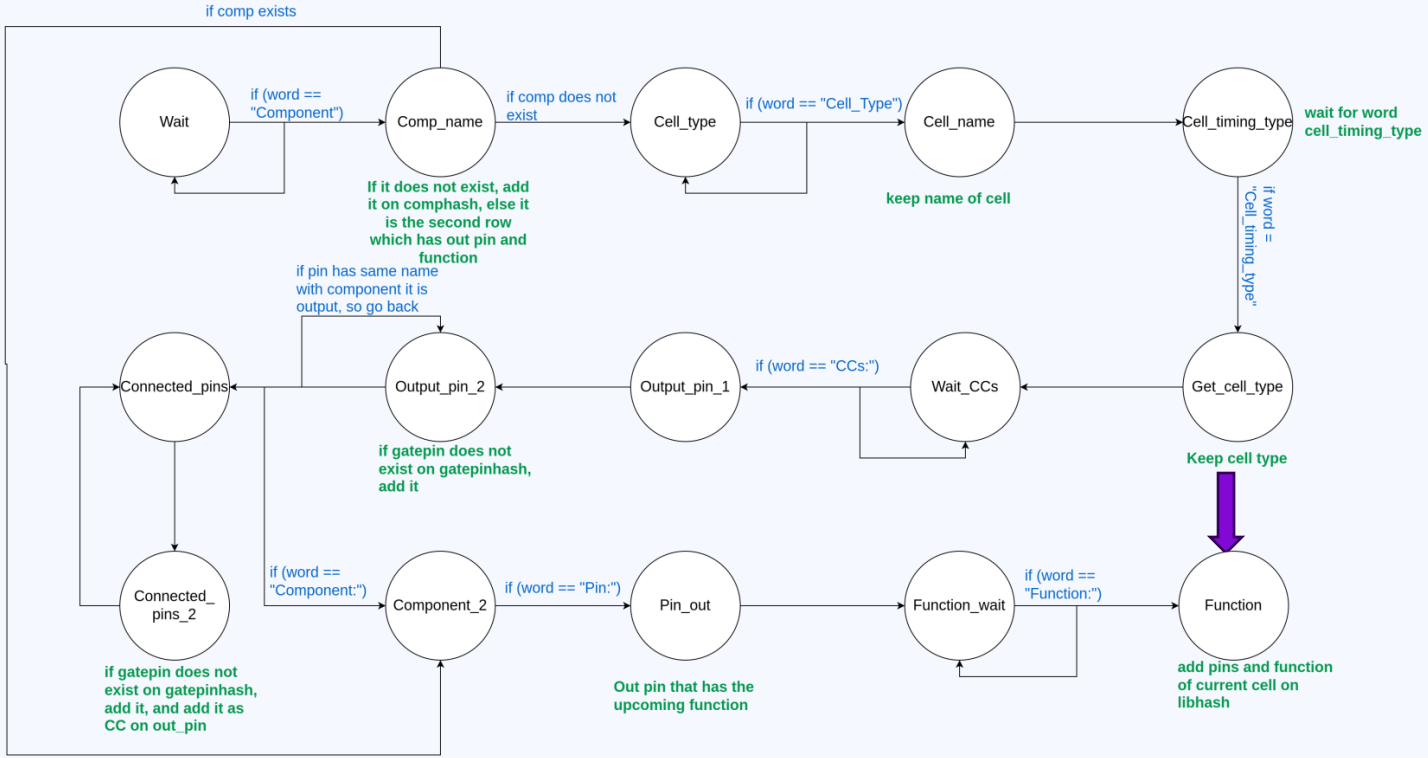
(A*B)+(A*CI)+(B*CI)

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: A^B^CI



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

Three_counters /U116/I1

Cell_pin (SM)

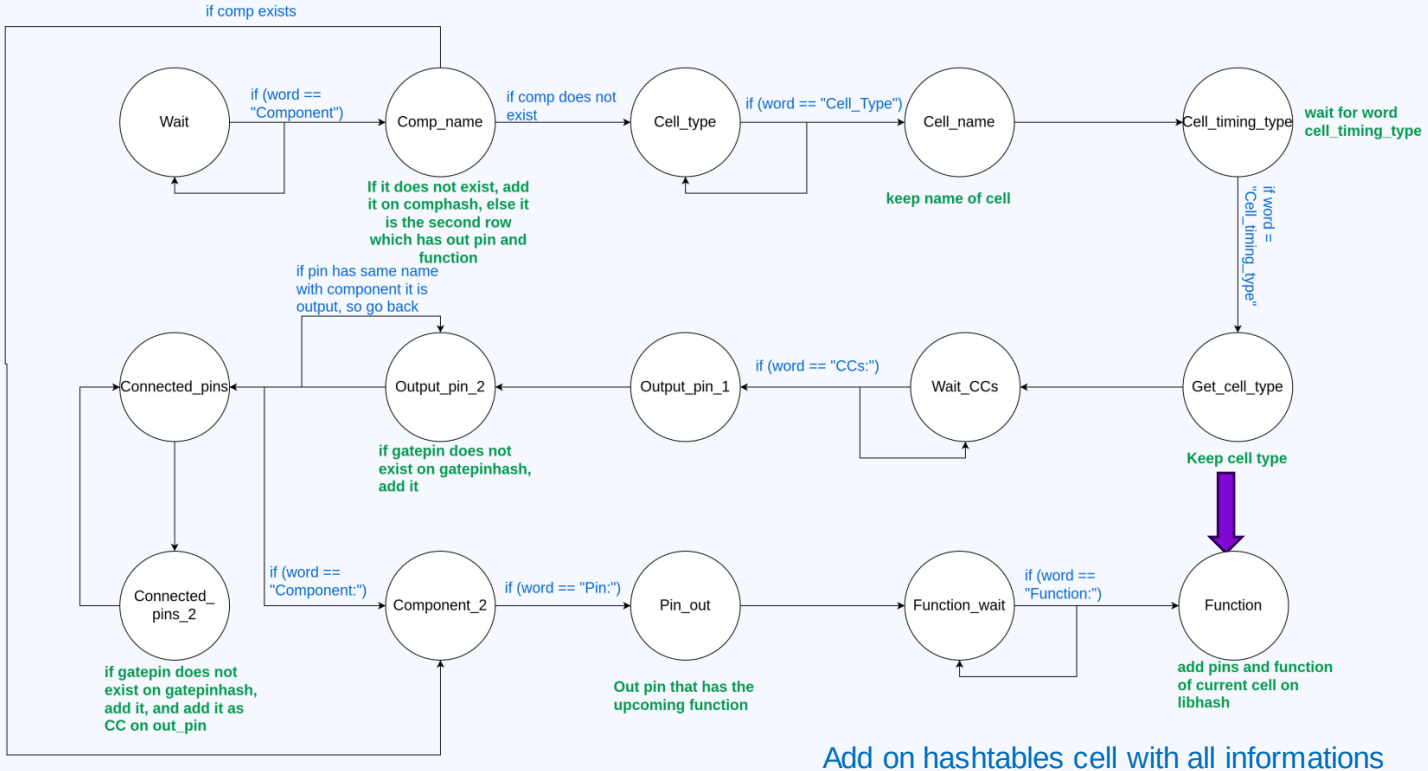
A^B^CI

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

Word: A^B^CI



Add on hashtables cell with all informations

Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

Three_counters /U29/SM

Three_counters /U116/I1

✓ Cell_pin (SM)

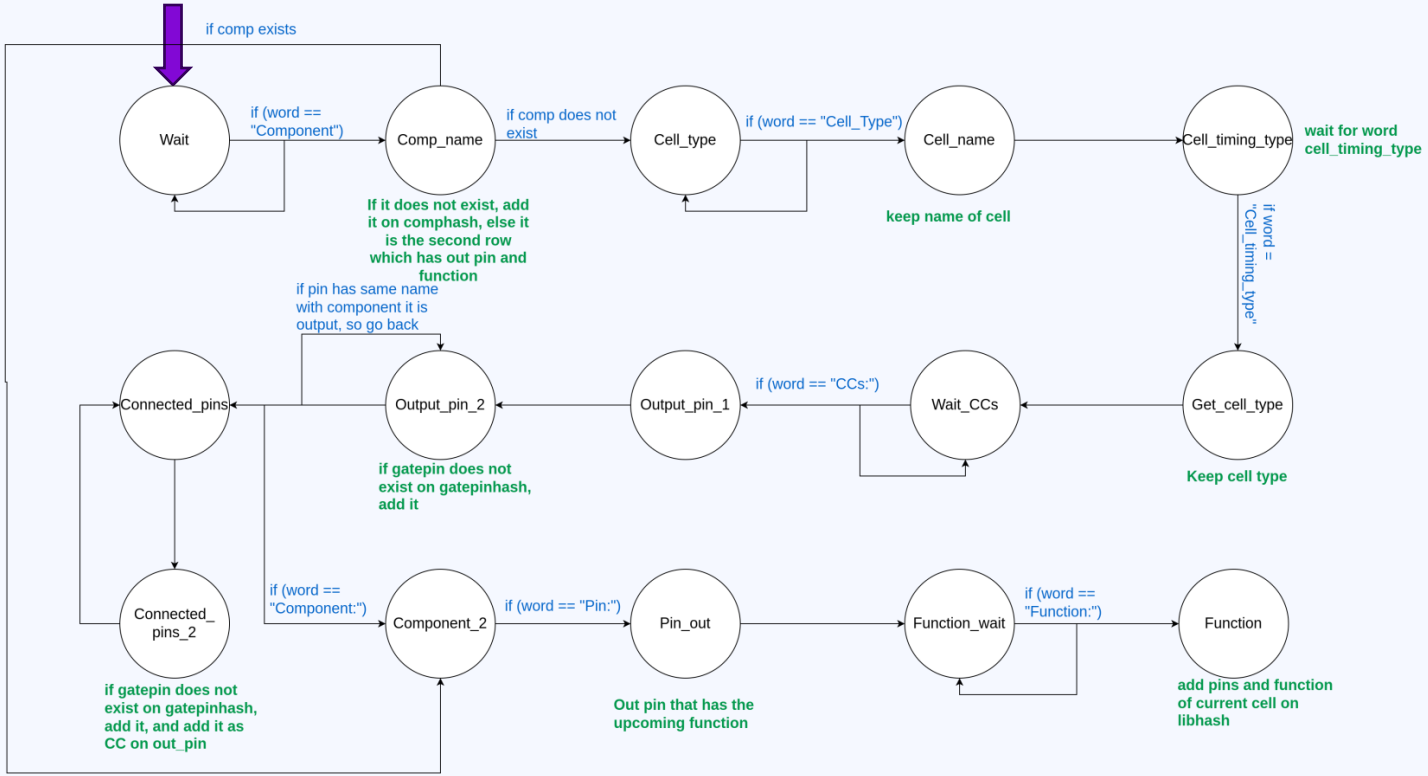
✓ A^B^CI

Component: three_counters/U29 Cell_Type: add2d1 Cell_Timing_Type: Combinational Width: 15.120 Height: 8.400 CCs: three_counters/U29 (/CO) three_counters/U30 (/CI) three_counters/U29 (/SM) three_counters/U116 (/I1)

Component: three_counters/U29, Output Pin: CO, Boolean Function: (A*B)+(A*CI)+(B*CI)

Component: three_counters/U29, Output Pin: SM, Boolean Function: A^B^CI

END



Comp_name

add2d1

Combinational

Three_counters /U29/CO

Three_counters /U30/CI

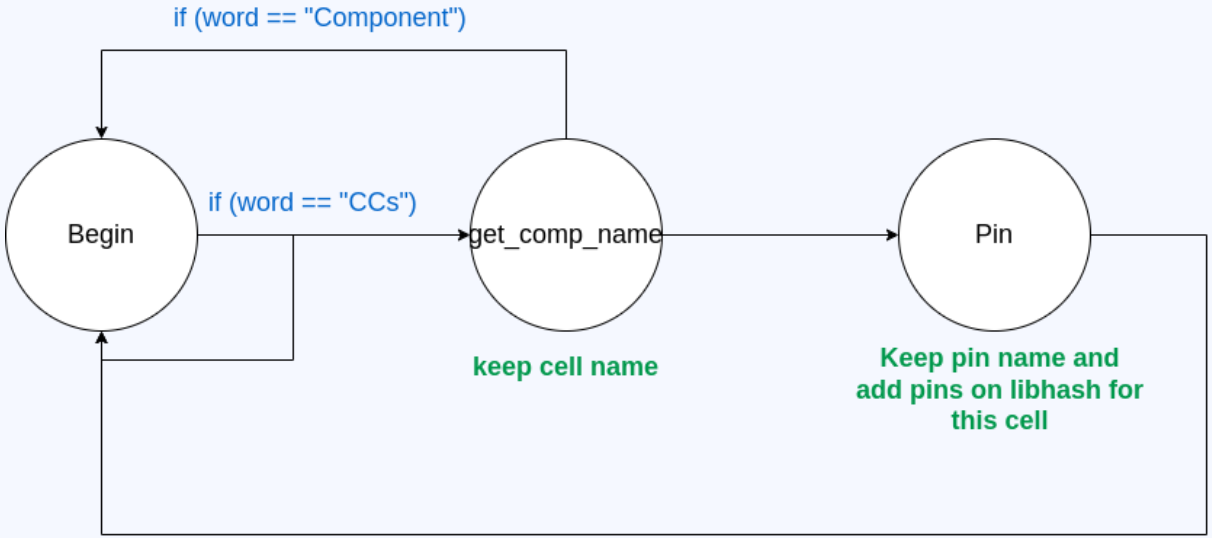
Three_counters /U29/SM

Three_counters /U116/I1

✓ Cell_pin (SM)

✓ A^B^CI

- Third parse: FSM that proccess 3rd part of file and search on gatepins for pins (input pins) to add them in libhash for current cell if they are not already exists



After Parsing

Now there are two fields that are empty:

- On gatepinsh the field that specifies which component they belong
- IOs type (if they are Input or Output)

We don't parse file again to fill these fields. We have already the hashtables with components and gatepins, so we take first part of gatepin name (which is component name) and we find hash and depth on comphash and pass them on gatepin

For IOs type, we get the from gatepinhash all IOs and check if first connection is Output or input. In case it has not connections we suppose that it is Input. In this way we fill this field.

```c
/* ################ gatepins_complete_parent() ################ */
/* This function adds the component that his pin belongs. This function
   runs when all structs are full, so it searches in gatepinhash it
   gets the pin name and it keeps the component name and then finds it
   in comphash and adds it as parent on gatepinhash
   For Example: gatepin is PID/U1288/Q we just keep PID/U1288 search
   it on comphash and get chash and cdepth and add it on gatepinhash*/
void gatepins_complete_parent()
{
    int i, j, k;
    char *comp_name = NULL;
    int chash, cdepth;

    for (i = 0; i < gatepinhash_size; i++)
    {
        for (j = 0; j < HASHDEPTH; j++)
        {
            if(gatepinhash[i].hashpresent[j] != 0  && gatepinhash[i].type[j] == WIRE)
            {
                for (k = strlen(gatepinhash[i].name[j]); gatepinhash[i].name[j][k] != '/'; k--)
                {

                }
                comp_name = (char *) calloc(k+1, sizeof(char));
                strncpy(comp_name, gatepinhash[i].name[j], k);
                get_comphash_indices(comp_name, &chash, &cdepth);

                free(comp_name);
                if(cdepth != -1)
                {
                    gatepinhash[i].parentComponent[j] = chash;
                    gatepinhash[i].parentComponentDepth[j] = cdepth;
                }
            }
        }
    }
}
```

Hash Function

Hash Function that returns the position of bucket according to size of hash table

```c
/* #### hash_function(const char *str, unsigned int num_buckets) #### */
/* This is a hash function that gets as input number of backets of
   one hash and it returns a key */
unsigned int hash_function(const char *str, unsigned int num_buckets)
{
    unsigned long hash_value = 5381; // Initial value

    while (*str != '\0')
    {
        hash_value = (hash_value * 33) ^ (unsigned long)(*str);
        str++;
    }

    // Use modulo to restrict the hash value to the range [0, num_buckets-1]
    return (unsigned int)(hash_value % num_buckets);
}
```

Structs Initialazation

```c
/* ##################### Gatepins_init() ##################### */
/* This function just initialize all fields of gatepin hash */
void Gatepins_init()
{
    int i, j;
    gatepinhash = (gatePins*) my_calloc(gatepinhash_size, sizeof(gatePins)); // size of 10 positions //

    for(i = 0; i < gatepinhash_size; i++)
    {
        for(j = 0; j < HASHDEPTH; j++) // what is the value of hashdepth
        {
            gatepinhash[i].name[j] = NULL;
            gatepinhash[i].pinConn[j] = NULL;    // no need i do this in add //
            gatepinhash[i].pinConnDepth[j] = NULL;

            gatepinhash[i].type[j] = -1;   // init type //
            gatepinhash[i].connections_size[j] = 1;
            gatepinhash[i].parentComponent[j] = -1;
            gatepinhash[i].parentComponentDepth[j] = -1;
            gatepinhash[i].hashpresent[j] = 0;
        }
    }      Dimitris Tsalapatas, last month • working all, i need to make the lists functions
}
```

```c
/* ##################### comphash_init() ##################### */
/* This function initializes the comphash data structure, which is used
   for storing components. It allocates memory and initializes each slot
   in the comphash with default values */
void comphash_init()
{
    int i, j;

    comphash = (Components*) my_calloc(comphash_size, sizeof(Components));
    for (i = 0; i < comphash_size; i++)
    {
        for (j = 0; j < HASHDEPTH; j++)
        {
            comphash[i].name[j] = NULL;
            comphash[i].lib_type[j] = -1;
            comphash[i].lib_type_depth[j] = -1;
            comphash[i].hashpresent[j] = 0;
        }
    }
}
```

```c
/* ######################## Lib_init() ######################## */
/* This function initializes all values for library hash table */
void Lib_init()      Dimitris Tsalapatas, last month • working all,
{
    int i, j;
    libhash = (Lib*) my_calloc(libhash_size, sizeof(Lib));

    for(i = 0; i < libhash_size; i++)
    {
        for(j = 0; j < HASHDEPTH; j++)
        {
            libhash[i].name[j] = NULL;
            libhash[i].function[j] = NULL;
            libhash[i].pin_names[j] = NULL;
            libhash[i].pin_count[j] = 0;
            libhash[i].hashpresent[j] = 0;
            libhash[i].cell_type[j] = -1;
            libhash[i].out_pins_count[j] = 0;
            libhash[i].pin_type[j] = NULL;
        }
    }
}
```

Functions for gatepinhash

- Gatepins_add

```c
/* ##### Gatepins_add(char *pin_name, int pin_type) ##### */
/* This function adds in gatepinhash a pin (its name and the type) */
void Gatepins_add(char *pin_name, int pin_type)
{
    int i;
    unsigned int key;

    if(pin_name == NULL)
        return;

    key = hash_function(pin_name, gatepinhash_size); // hash again string to get same key //

    for(i = 0; i < HASHDEPTH; i++)
    {
        if(gatepinhash[key].hashpresent[i] == 0)    // empty //
            break;
    }
    gatepinhash[key].name[i] = (char *) my_calloc(1, (strlen(pin_name) + 1) );

    strcpy(gatepinhash[key].name[i], pin_name);

    /* Calloc for the new pin size for each connections, 1 for begin and go on*/
    gatepinhash[key].pinConn[i] = (int *) my_calloc(1, sizeof(int));
    gatepinhash[key].pinConnDepth[i] = (int *) my_calloc(1, sizeof(int));

    gatepinhash[key].connections_size[i] = 0;
    gatepinhash[key].type[i] = pin_type;
    gatepinhash[key].hashpresent[i] = 1;

    #ifdef DEBUG
    printf("Gatepin inserted succesfully\n");
    #endif
}
```

- Get_gatepin_indices

```c
/* ## get_gatepin_indices(char *pin_name, int *ghash, int *ghashdepth) ### */
/* This function search gatepinhash for this pin and returns value
   of position in ghash and ghashdepth. If it does not find it
   it returns ghashdepth = -1 */
void get_gatepin_indices(char *pin_name, int *ghash, int *ghashdepth)
{
    int i;
    unsigned int key;
    key = hash_function(pin_name, gatepinhash_size);
    *ghash = key;
    *ghashdepth = -1;   // add this value if it does not find pin //

    for(i = 0; i < HASHDEPTH; i++)
    {
        if(gatepinhash[*ghash].name[i] != NULL)
        {
            if(strcmp(gatepinhash[*ghash].name[i], pin_name) == 0)
            {
                *ghashdepth = i;
                break;
            }
        }
    }
}
```

Functions for gatepinhash

- gatepin_add_CCs

```c
/* #### gatepin_add_CCs(char *source_pin, char *connection_pin) #### */
/* This function adds connections for each gatepin. It keeps an array
   of gatepinhash and gatepinhashdepth which are positions in gatepinhash */     You, last month • some comments on desgin
void gatepin_add_CCs(char *source_pin, char *connection_pin)
{
    int source_hash, source_hash_depth;
    int connection_hash, connection_hash_depth;
    int size_of_connections = 0;

    if(source_pin == NULL || connection_pin == NULL)
        return;

    get_gatepin_indices(source_pin, &source_hash, &source_hash_depth);
    get_gatepin_indices(connection_pin, &connection_hash, &connection_hash_depth);

    if(source_hash_depth == -1 || connection_hash_depth == -1)
    {
        printf("Source or Connection pin does not exists!\n");
        return;
    }

    if(gatepinhash[source_hash].pinConnDepth[source_hash_depth] == NULL)
    {
        printf("Invalid pointer\n");
    }

    gatepinhash[source_hash].pinConn[source_hash_depth] = (int*) my_realloc(gatepinhash[source_hash].pinConn[source_hash_depth],
    gatepinhash[source_hash].pinConnDepth[source_hash_depth] = (int*) my_realloc(gatepinhash[source_hash].pinConnDepth[source_has

    size_of_connections = gatepinhash[source_hash].connections_size[source_hash_depth]; // get size of connections for current pi

    gatepinhash[source_hash].pinConn[source_hash_depth][size_of_connections] = connection_hash;
    gatepinhash[source_hash].pinConnDepth[source_hash_depth][size_of_connections] = connection_hash_depth;

    gatepinhash[source_hash].connections_size[source_hash_depth]++;
}
```

- gatepins_complete_parent

```c
/* ################ gatepins_complete_parent() ################ */
/* This function adds the component that his pin belongs. This function
   runs when all structs are full, so it searches in gatepinhash it
   gets the pin name and it keeps the component name and then finds it
   in comphash and adds it as parent on gatepinhash
   For Example: gatepin is PID/U1288/Q we just keep PID/U1288 search
   it on comphash and get chash and cdepth and add it on gatepinhash*/
void gatepins_complete_parent()
{
    int i, j, k;
    char *comp_name = NULL;
    int chash, cdepth;

    for (i = 0; i < gatepinhash_size; i++)
    {
        for (j = 0; j < HASHDEPTH; j++)
        {
            if(gatepinhash[i].hashpresent[j] != 0  && gatepinhash[i].type[j] == WIRE)
            {
                for (k = strlen(gatepinhash[i].name[j]); gatepinhash[i].name[j][k] != '/'; k--)
                {

                }
                comp_name = (char *) calloc(k+1, sizeof(char));
                strncpy(comp_name, gatepinhash[i].name[j], k);
                get_comphash_indices(comp_name, &chash, &cdepth);

                free(comp_name);
                if(cdepth != -1)
                {
                    gatepinhash[i].parentComponent[j] = chash;
                    gatepinhash[i].parentComponentDepth[j] = cdepth;
                }
            }
        }
    }
}
```

Functions for Libhash

- Lib_add

```c
/* ## Lib_add(char *cell_name, int cell_type, char *func_expr) ## */
/* This function adds in libhash for each cell the name of
   cell the cell_type (Combinational or Sequential) */      You, 1 second ago • Uncommitted changes
void Lib_add(char *cell_name, int cell_type)
{
    int i;
    unsigned int key;
    int lhash, ldepth;

    if(cell_name == NULL)
        return;

    key = hash_function(cell_name, libhash_size); // rehash cell_name to get pos of it in libhash //

    get_libhash_indices(cell_name, &lhash, &ldepth);
    if(ldepth == -1)
    {
        for (i = 0; i < HASHDEPTH; i++)
        {
            if(libhash[key].hashpresent[i] == 0)
                break;
        }
        libhash[key].name[i] = (char *) my_calloc((strlen(cell_name) + 1), sizeof(char));
        strcpy(libhash[key].name[i], cell_name);

        libhash[key].cell_type[i] = cell_type;
        libhash[key].hashpresent[i] = 1;

        ldepth = i;
    }
    #ifdef DEBUG
    printf("Cell inserted succesfully on libhash\n");
    #endif
}
```

- get_libhash_indices

```c
/* ## get_libhash_indices(char *cell_name, int *lhash, int *lhashdepth) ## */
/* This function retrieves the hash index and depth for a given cell_name
   in the libhash data structure. The hash index represents the position
   in libhash where information about the specified cell is stored, and
   the depth represents the specific slot within that position. */
void get_libhash_indices(char *cell_name, int *lhash, int *lhashdepth)
{
    int i;
    unsigned int key;
    key = hash_function(cell_name, libhash_size);
    *lhash = key;

    *lhashdepth = -1; // initialize depth to -1, indicating cell not found in libhash //

    for(i = 0; i < HASHDEPTH; i++)
    {
        if(libhash[*lhash].hashpresent[i] != 0)
        {
            if(strcmp(libhash[*lhash].name[i], cell_name) == 0)
            {
                *lhashdepth = i;
                break;
            }
        }
    }
}
```

Functions for Libhash

- Lib_add_pins

```
/* ######### lib_add_pins(char *cell_name, char *pin_name) ######### */
/* This function adds pin information to the libhash data structure for
   a specified cell. It checks for duplicate pin names and ensures that
   the pin information is appropriately stored in the libhash. */
void lib_add_pins (char *cell_name, char *pin_name, int pin_type)
{
    int i;
    int lhash, ldepth;

    if(cell_name == NULL)
    {
        return;
    }

    // Retrieve the hash index and depth for the specified cell_name //
    get_libhash_indices(cell_name, &lhash, &ldepth);
    if(ldepth == -1)
    {
        printf("ERROR: There is not this cell\n");
        return;
    }

     // Check for duplicate pin names in the specified cell //
    for(i = 0; i < libhash[lhash].pin_count[ldepth]; i++)
    {
        if(libhash[lhash].pin_names[ldepth][i] != NULL)
        {
            if (strcmp(libhash[lhash].pin_names[ldepth][i], pin_name) == 0 )
            {
                return; // already exists //
            }
        }
    }

    // Add the new pin name at position i //
    libhash[lhash].pin_count[ldepth]++ ;
    libhash[lhash].pin_names[ldepth] = (char **) realloc(libhash[lhash].pin_names[ldepth], (libhash[lhash].pin_count[ldepth]) * sizeof(char *));
    libhash[lhash].pin_names[ldepth][libhash[lhash].pin_count[ldepth] - 1] = (char *) calloc(strlen(pin_name) + 1, sizeof(char));

    strcpy(libhash[lhash].pin_names[ldepth][libhash[lhash].pin_count[ldepth] - 1], pin_name);

    libhash[lhash].pin_type[ldepth] = (int *) realloc(libhash[lhash].pin_type[ldepth], (libhash[lhash].pin_count[ldepth]) * sizeof(int));
    libhash[lhash].pin_type[ldepth][libhash[lhash].pin_count[ldepth] - 1] = pin_type;
}
```

- Lib_add_func

```
/* ######### lib_add_func(char *cell_name, char *func_expr) ######### */
/* This function adds function to libhash structure for
   a specified cell. It checks for duplicate functions and ensures that
   the function information is appropriately stored in the libhash. */
void lib_add_func(char *cell_name, char *func_expr)
{
    int i;
    int lhash, ldepth;

    get_libhash_indices(cell_name, &lhash, &ldepth);
    if(ldepth == -1)
        return;

    for (i = 0; i < libhash[lhash].out_pins_count[ldepth]; i++)
    {
        if ( strcmp(libhash[lhash].function[ldepth][i], func_expr) == 0)
        {
            return;
        }
    }

    libhash[lhash].out_pins_count[ldepth]++;
    libhash[lhash].function[ldepth] = (char **) my_realloc(libhash[lhash].function[ldepth], sizeof(char*) *
    libhash[lhash].function[ldepth][i] = (char *) calloc(strlen(func_expr) + 1, sizeof(char));
    strcpy(libhash[lhash].function[ldepth][i], func_expr);
}
```

Functions for Comphash

- Comphash_add

```
/* ## comphash_add(char *comp_name, char *cell_name, int cell_type, char *func_expr) ## */
/* This function adds component information to the comphash data structure.
   It associates a component name with a cell from the libhash, storing
   additional information like the libhash indices and depth. If the
   cell does not exist in libhash, it adds the cell using Lib_add function. */
void comphash_add(char *comp_name, char *cell_name, int cell_type, char *func_expr)
{
    int i;
    unsigned int key;
    int lhash, ldepth;

    if (comp_name == NULL)
    {
        return;
    }
    key = hash_function(comp_name, comphash_size);

    for (i = 0; i < HASHDEPTH; i++)
    {
        if(comphash[key].hashpresent[i] == 0) // slot is empty //
        {
            break;
        }
    }

    if(i == HASHDEPTH)
    {
        printf("No space!\n");
        return; // no space available in comphash //
    }

    // Allocate memory for the component name and copy it to comphash //
    comphash[key].name[i] = (char *) calloc(strlen(comp_name) + 1, sizeof(char));
    strcpy(comphash[key].name[i], comp_name);

    comphash[key].hashpresent[i] = 1;
    #ifdef DEBUG
    printf("Comp inserted succesfully\n");
    #endif
```

```
    // Check if the cell already exists in libhash //
    get_libhash_indices(cell_name, &lhash, &ldepth);
    if(ldepth != -1)
    {
        comphash[key].lib_type[i] = lhash;
        comphash[key].lib_type_depth[i] = ldepth;

        return;
    }

    // If the cell does not exist in libhash, add it using Lib_add function //
    Lib_add(cell_name, cell_type, func_expr);
    get_libhash_indices(cell_name, &lhash, &ldepth);

    // Update the lib_type and lib_type_depth in comphash //
    comphash[key].lib_type[i] = lhash;
    comphash[key].lib_type_depth[i] = ldepth;
}
```

Structs Free

```c
/* ######################## comphash_free() ######################## */
/* This function frees the memory allocated for the comphash data structure,
   including component names, within each slot of the comphash. */
void comphash_free()
{
    int i, j;

    for(i = 0; i < comphash_size; i++)
    {
        for(j = 0; j < HASHDEPTH; j++)
        {
            if(comphash[i].hashpresent[j] != 0)
            {
                free(comphash[i].name[j]);
            }
        }
    }
    free(comphash);
}
```

```c
/* ######################## libhash_free() ######################## */
/* This function frees the memory allocated for the libhash data structure,
   including names, functions, and pin information for each cell */
void libhash_free()      Dimitris Tsalapatas, last month • working all, i need to make the lists functions
{
    int i, j, k;

    for(i = 0; i < libhash_size; i++)
    {
        for(j = 0; j < HASHDEPTH; j++)
        {
            if(libhash[i].hashpresent[j] != 0) // Check if the slot is occupied //
            {
                free(libhash[i].name[j]);
                for(k = 0; k < libhash[i].out_pins_count[j]; k++)
                {
                    free(libhash[i].function[j][k]);
                }
                free(libhash[i].function[j]);

                for(k = 0; k < libhash[i].pin_count[j]; k++) // Iterate through each pin in the current cell //
                {
                    free(libhash[i].pin_names[j][k]);
                }
                free(libhash[i].pin_names[j]);
                free(libhash[i].pin_type[j]);
            }
        }
    }
    free(libhash); // Free the memory allocated for the entire libhash data structure //
}
```

```c
/* ##################### Gatepins_free() ##################### */
/* This function free all fields of this struct */
void Gatepins_free()      Dimitris Tsalapatas, last month • work
{
    int i, j;

    for(i = 0; i < gatepinhash_size; i++)
    {
        for(j = 0; j < HASHDEPTH; j++)
        {
            if(gatepinhash[i].name[j] != NULL)
            {
                free(gatepinhash[i].name[j]);
                free (gatepinhash[i].pinConn[j]);
                free (gatepinhash[i].pinConnDepth[j]);
            }
        }
    }
    free(gatepinhash);
}
```

Supported TCL commands

- **Read_design:** Read current file
- **List_IOs:** List all IOs of design and their connections
- **List_components:** print a TCL list with all components of design
- **Report_component_function:** Print functions for each output pin of given component
- **Report_component_type:** Print type of component (Combinational or Sequential)
- **List_component_CCS:** Print connections of current component
- **List_IO_CCS:** Print connections of current IO
- **List_cell:** Print all fields of given cell
- **List_cells:** Print all fields of all cells of design
- **List_component_info:** Print cell type and connections of given component
- **List_components_info:** Print cell type and connections of all components
- **Clear_design:** Clear and free all structures

Some Examples

```c
int list_IO(ClientData clientdata, Tcl_Interp *interp, int objc, Tcl_Obj *const* objv)
{
    int i, j;
    Tcl_Obj *currPin, *result_pins;

    result_pins = Tcl_NewListObj(0, NULL);
    if(gatepinhash == NULL)
    {
        printf(ANSI_COLOR_RED "ERROR: No design loaded" ANSI_COLOR_RESET);      Dimitris Tsalapatas, las
        return TCL_ERROR;
    }

    for (i = 0; i < gatepinhash_size; i++)
    {
        for(j = 0; j < HASHDEPTH; j++)
        {
            if(gatepinhash[i].hashpresent[j] != 0)
            {
                if(gatepinhash[i].type[j] == IO_TYPE || gatepinhash[i].type[j] == PO)
                {
                    currPin = Tcl_NewStringObj(gatepinhash[i].name[j], strlen(gatepinhash[i].name[j]));

                    Tcl_ListObjAppendElement(interp, result_pins, currPin);
                }
            }
        }
    }
    Tcl_SetObjResult(interp, result_pins);

    return TCL_OK;
}
```

```c
int list_components(ClientData clientdata, Tcl_Interp *interp, int objc, Tcl_Obj *const* objv)
{
    int i, j;
    Tcl_Obj *currComp, *result_comps;

    result_comps = Tcl_NewListObj(0, NULL);
    if(comphash == NULL)
    {
        printf(ANSI_COLOR_RED "ERROR: No design loaded" ANSI_COLOR_RESET);
        return TCL_ERROR;
    }

    for (i = 0; i < comphash_size; i++)
    {
        for(j = 0; j < HASHDEPTH; j++)
        {
            if(comphash[i].hashpresent[j] != 0)
            {
                currComp = Tcl_NewStringObj(comphash[i].name[j], strlen(comphash[i].name[j]));

                Tcl_ListObjAppendElement(interp, result_comps, currComp);
            }
        }
    }
    Tcl_SetObjResult(interp, result_comps);

    return TCL_OK;
}
```

Some Examples

```
int list_cells(ClientData clientdata, Tcl_Interp *interp, int objc, Tcl_Obj *const* objv)
{
    int i;
    int lhash, ldepth;

    if(objc != 1)
    {
        Tcl_WrongNumArgs(interp, 1, objv, "no argument");
        return TCL_ERROR;
    }       You, 2 weeks ago • new TCL commands added to be easier to identify c…

    if(gatepinhash == NULL)
    {
        printf(ANSI_COLOR_RED "ERROR: No design loaded" ANSI_COLOR_RESET);
        return TCL_ERROR;
    }

    for(lhash = 0; lhash < libhash_size; lhash++)
    {
        for(ldepth = 0; ldepth < HASHDEPTH; ldepth++)
        {
            if(libhash[lhash].hashpresent[ldepth] == 1)
            {
                printf(ANSI_COLOR_BLUE"------------ INFO CELL: %s ------------\n" ANSI_COLOR_RESET, libhash[lhash].name[ldepth]);
                if(libhash[lhash].cell_type[ldepth] == COMBINATIONAL)
                {
                    printf(ANSI_COLOR_ORANGE"Cell Type is: Combinational\n" ANSI_COLOR_RESET);
                }
                else
                {
                    printf(ANSI_COLOR_MAGENDA "Cell Type is: Sequential\n" ANSI_COLOR_RESET);
                }

                printf("Pin names are: \n");
                for(i = 0; i < libhash[lhash].pin_count[ldepth]; i++)
                {
                    printf(ANSI_COLOR_GREEN "%d) %s " ANSI_COLOR_RESET, i+1, (libhash[lhash].pin_names[ldepth][i] + 1));
                    if(libhash[lhash].pin_type[ldepth][i] == OUTPUT)
                    {
                        printf(ANSI_COLOR_GREEN "output pin with function %s\n" ANSI_COLOR_RESET, libhash[lhash].function[ldepth][i]);
                    }
                    else
                    {
                        printf(ANSI_COLOR_GREEN "input pin\n" ANSI_COLOR_RESET);
                    }
                }

                printf(ANSI_COLOR_BLUE "-------------------------------------------\n\n" ANSI_COLOR_RESET);
            }
        }
    }

    return TCL_OK;
}
```

Extras

Custom functions for realloc and calloc to don't check return value every time

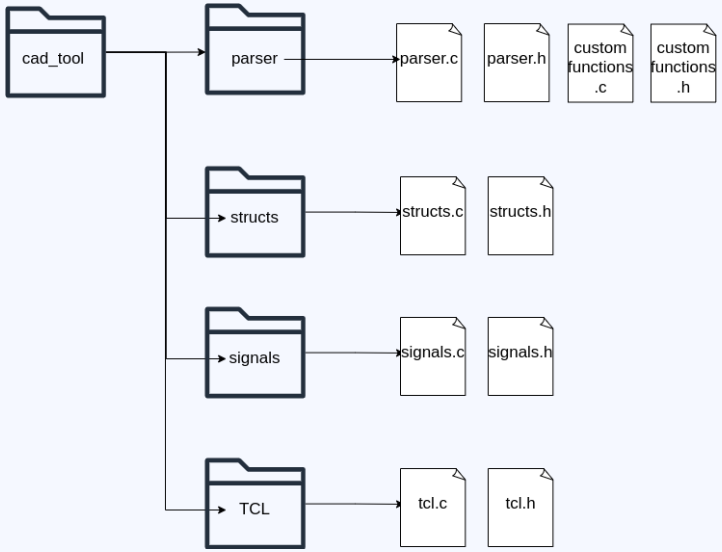
```c
void* my_realloc(void* ptr, size_t size)
{
    ptr = realloc(ptr, size);
    if(ptr == NULL)
    {
        printf("Error on allocation\n");

        exit(EXIT_FAILURE);
    }
    return ptr;
}

void* my_calloc(int nmemb, size_t size)
{
    void *ptr;
    ptr = calloc(nmemb, size);
    if(ptr == NULL)
    {
        printf("Error on allocation\n");

        exit(EXIT_FAILURE);
    }
    return ptr;
}
```

Signal handler for CTRL+C (interrupt program) to ask user if really wants to exit

```c
void sigint_handler(int signum)
{
    printf(ANSI_COLOR_ORANGE "\nCTRL+C pressed. Do you want to exit? (y/n): " ANSI_COLOR_RESET);
    char response;
    scanf(" %c", &response);

    while (1)
    {
        if (response == 'y' || response == 'Y')
        {
            exit_requested = true;
            ctrl_c_pressed = true;
            break;
        }
        else if (response == 'n' || response == 'N')
        {
            ctrl_c_pressed = false;
            printf("Continuing...\nPR> ");
            break;
        }
        else
        {
            printf(ANSI_COLOR_RED "Incorrect answer! Please enter 'y' or 'n': " ANSI_COLOR_RESET);
            // Clear input buffer
            ctrl_c_pressed = false;
            while (getchar() != '\n');
            scanf(" %c", &response);
        }
    }
}
```

Makefile

```
CXX = gcc
CXXFLAGS = -Wall -g
LDFLAGS = -lm -ltcl -lreadline -lcudd
SRC_DIRS = structs parser tcl CUDD_impl convert_infix signals
SRC_FILES = $(foreach dir,$(SRC_DIRS),$(wildcard $(dir)/*.c))
OBJ_FILES = $(SRC_FILES:.c=.o)
TARGET = cad_tool

# Main rule
all: $(TARGET)

# Compile source files into object files
%.o: %.c
    $(CXX) $(CXXFLAGS) -c $< -o $@

# Link object files to create the executable
$(TARGET): $(OBJ_FILES)
    $(CXX) $^ -o $@ $(LDFLAGS)

# Clean compiled files
clean:
    rm -f $(OBJ_FILES) $(TARGET)

.PHONY: all clean
```



Thank you for your attention!

Do you have any questions?