ECE556 Project
Part 1

Azadeh Davoodi

Part 1: Overview

- In this part you implement a “simple” router and the main goal is to become familiar with the data structures and the input/output formats
- You should fill in the template source code (download from the project website) and modify it to allow creating an executable called “ROUTE.exe” with the exact format and input arguments given below:

  ./ROUTE.exe <input_benchmark_name> <output_file_name>

- Your output file should then pass the evaluation script posted on the project website (with input argument “eval_mode=0”)

Template Source Code

- Contains 4 files
  1. main.cpp
  2. ece556.h
  3. ece556.cpp
  4. Makefile
- You only need to implement some functions (explained in the next few slides) which are all located inside ece556.cpp
- Your code will be compiled and evaluated on the CAE tux machines so make sure it is compatible with those machines
- Use of existing data structures (described in ece556.h) strongly recommended
  - OK to change data structures, but any problems occurred (e.g., running out of memory or runtime getting too slow) in this part or future parts will be your responsibility

Overview of main.cpp

```
// Read input benchmark
        status = readBenchmark(inputFileName, rst);
        if(status==0){
                printf("ERROR: reading input file \n"); return 1;
        }
// run actual routing
        status = solveRouting(rst);
        if(status==0){
                printf("ERROR: running routing \n");release(rst);return 1;
        }
// write output routing file
        status = writeOutput(outputFileName, rst);
        if(status==0){
                printf("ERROR: writing the result \n"); release(rst); return 1;
        }
// release memory
        release(rst);
```

Part 1: Tasks

- Fill in the following functions in the template source code (ece556.cpp)
  1. int readBenchmark(const char *fileName, routingInst *rst);
  2. int solveRouting(routingInst *rst);
  3. int writeOutput
     (const char *outRouteFile, routingInst *rst);
  4. int release(routingInst *rst);
- Your output file should successfully pass the evaluation script posted on the project website
  - with input argument eval_mode=0 indicating a longer evaluation mode that checks the validity of the routing solution (i.e., all pins of a net are indeed connected to each in your generated solution, and ensuring that the pin locations have not been altered in the generate solution compared to the provided input benchmark file)

1) int readBenchmark (const char *fileName, routingInst *rst);

- Reads in the benchmark file and initializes the routing instance
  - This function needs to populate some fields in the routingInst structure
  - Refer to the project website for description of the input format
  - input1: fileName: Name of the benchmark input file
  - input2: pointer to the routing instance
  - output: 1 if successful

2) int solveRouting(routingInst *rst)

- This function creates a routing solution
  - input: pointer to the routing instance
  - output: 1 if successful, 0 otherwise (e.g. if the data structures are not populated)
- In this part of the project, we will implement a simple router inside the solveRouting function which is explained in the future slides

3) int writeOutput (const char *outRouteFile, routingInst *rst);

- Write the routing solution obtained from solveRouting().
  - input1: name of the output file
  - input2: pointer to the routing instance
  - output: 1 if successful, 0 otherwise (e.g. the output file is not valid)
- Refer to the project website for the required output format.
- Make sure your generated output file passes the evaluation script to verify it is in the correct format and the nets have been correctly routed. The script also reports two quality metrics: (1) total wirelength and (2) total edge overflow, for your routing solution.

4) int release(routingInst *rst);

- Releases the memory for all the allocated data structures
  - input: pointer to the routing instance
  - output: 1 if successful, 0 otherwise
- Failure to release may cause memory problems after multiple runs of your program
- Need to recursively delete all memory allocations from bottom to top (e.g., starting from segments then routes then individual fields within a net struct, then the nets, then the fields in a routing instance, and finally the routing instance)

Overview of Data Structures

- point: A structure to represent a 2D Point

```
typedef struct
{
  int x ;   /* x coordinate ( >=0 in the routing grid)*/
  int y ;   /* y coordinate ( >=0 in the routing grid)*/

} point ;
```

- segment: A structure to represent a sequence of connected edges (does not have to be a flat line)

```
typedef struct
{
  point p1 ;       /* start point of a segment */
  point p2 ;       /* end point of a segment */

  int numEdges ; /* number of edges in the segment*/
  int *edges ;     /* array of edge indexes representing a (continuous) path*/

} segment ;
```

- route: A structure to represent a route

```
typedef struct
 {
   int numSegs ;                /* number of segments in a route*/
   segment *segments ;          /* an array of segments (note, a segment may have any shape, for example flat, L-shaped, etc.)

 } route ;
```

Overview of Data Structures

- Net: A structure to represent the connectivity and route of an interconnect

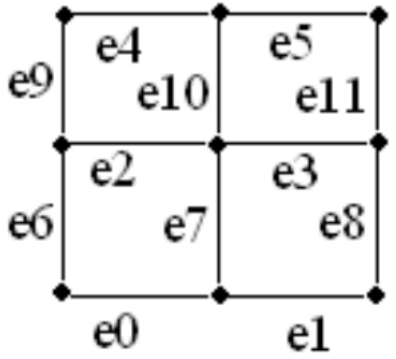
```
typedef struct
 {

  int id ;                  /* ID of the net */
  int numPins ;             /* number of pins (or terminals) of the net */
  point *pins ;             /* array of pins (or terminals) of the net. */
  route nroute ;            /* stored route for the net. */

 } net ;
```

- routingInst: A structure to represent the routing instance

```
typedef struct
 {
  int gx ;                  /* x dimension of the global routing grid */
  int gy ;                  /* y dimension of the global routing grid */

  int cap ;                 /* default capacity of the edges */

  int numNets ;             /* number of nets */
  net *nets ;               /* array of nets */

  int numEdges ;            /* number of edges of the grid */
  int *edgeCaps;            /* array of the actual edge capacities after adjustment for blockages */
  int *edgeUtils;           /* array of edge utilizations */

 } routingInst ;
```

Number of Edges in A Grid

- How many edges are in a grid of granularity X, Y?
- #edges = [Y*(X-1) + X*(Y-1)]
- Example: For a grid with X=3 and Y=3 we have 12 edges as shown in the figure in the 2-D projected graph.
- Please do not assume the grid is square (i.e., X and Y dimensions are equal)



int solveRouting(routingInst *rst)

- Simple ideas on implementing this function using the provided data structures
  1. View the array of pins stored for a net as union of two-terminal sub-nets
     - For example: net->pins: p1 → p2 → p3 → p5 is viewed as the union of the following two-terminal sub-nets: p1→p2, p2→ p3, p3→p5
  2. In the net data structure, each "segment" corresponds to a two-pin sub-net
     - If the sub-net is flat (i.e., the two pins have same x or same y), route as a flat line
     - If the sub-net is not flat, route the sub-net as L-shaped pattern
     - In the above cases, "routing" means storing the integer edge IDs composing the segment (segment->edges)
     - The above rules are used to create a simple routing solution for the first part of the project which can be used as initial solution to the second part of the project.

On Storing Edge IDs

- Can calculate a formula to *map* two given end points of an edge to a unique ID : edgeID = f(p1, p2)
- Can you guess the formula used for labeling the edge in the example in slide 12?
- Note, also need the reverse of the function (f^{-1}) to map edgeID to p1 and p2 in order to describe the route in the required format in the output file