Embedded Systems – Design Verification and Test

Dr. Santosh Biswas
Prof. Jatindra Kumar Deka
Dr. Arnab Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture – 09
Hardware Architectural Synthesis – 4

Welcome to module 1 of lecture 7. In the last module, we took a look at when are 2 operations said to be compatible, we also understood two important resource sharing models, the compatibility graph model and the conflict graph model. And we saw that a solution to compatibility graph model in terms of finding the minimum number of cliques in it. A minimum clique partition of the compatibility graph gives me the maximum resource sharing possible.

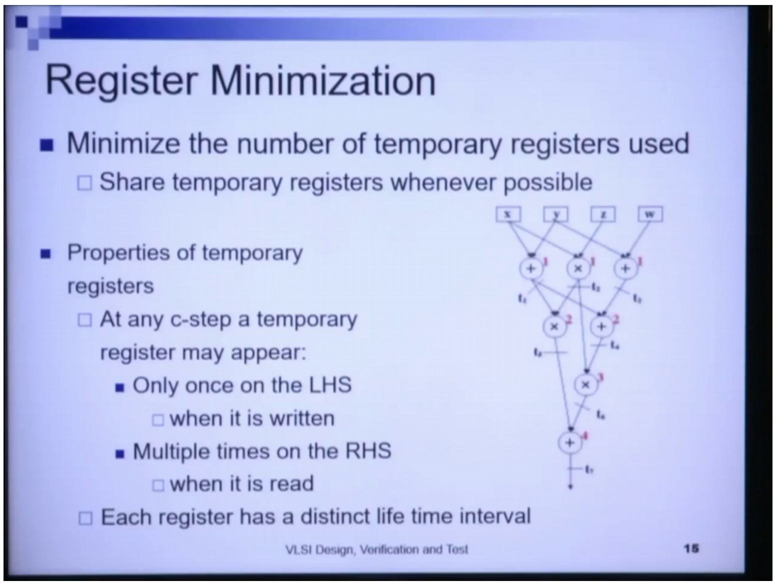
We also saw that minimum coloring of the conflict graph gives me an optimal to again to a resource sharing problem, but we told that both the minimum clique partitioning problem and the minimum graph coloring problem is NP complete for general graphs. Then we understood that there are a few special classes of graphs that are chordal graphs, interval graphs and compatibility graphs for which these problems have polynomial type optimal solutions.

We understood that the compatibility graph corresponding to the operation within simple data flow graph within a basic block can be modelled as an interval graph, why? Because each operation each an operation within a single constraints graph within a basic graph can be modelled as single continuous intervals from the start of it is execution to the end of it is execution.

And hence, the corresponding conflict and compatibility graph with respect to this scheduled operation constraint graph can be modelled as interval graphs. As we said, next after understanding the resource sharing models we will study algorithms for resource allocation. We said that interval graphs gives as easy algorithms for graph coloring in polynomial time and these are all optimal solutions. We will study one of these algorithms in this in this lecture. But before that we will understand the resource minimization problem.

We will understand the resource minimization problem can also be mapped as eh as graph coloring over interval graphs. And hence the same graph coloring algorithm can be used both for functional unit minimization; that means, maximization of resource sharing maximization of resource sharing among functional units, and maximization of resource sharing among registers as well. And hence, before going to the actual coloring for interval graphs and going to the actual algorithm for coloring interval graph; we will first understand the register minimization problem.

(Refer Slide Time: 03:51)



Register Minimization

- Minimize the number of temporary registers used
  - Share temporary registers whenever possible
- Properties of temporary registers
  - At any c-step a temporary register may appear:
    - Only once on the LHS
      - when it is written
    - Multiple times on the RHS
      - when it is read
  - Each register has a distinct life time interval

VLSI Design, Verification and Test 15

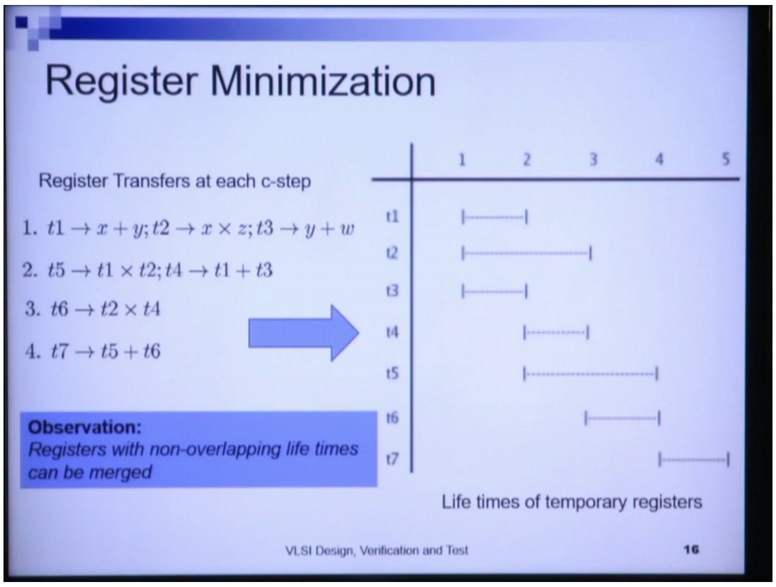
The problem to minimize the number of temporary registers used; so, we see that there are 4 independent variables here x, y, z, w. And 4 registers will any now 4 hardware register will anyhow be required to store this variables. So, for this operation to be scheduled in time step one, before time step one is activated x, y, z, w must be in different hardware registers for this operations to execute correctly. Now that is out of our control; however, these temporary registers, we said that the output of the operation is are floated on temporary registers.

So, each operation is floated on a different temporary registers at the output of the scheduling step. Now, the register minimization problem is to minimize the number of hardware registers that are required to allocate these temporary registers. So, these temporary registers must be allocated to hardware registers as well, right? When I in actually implement the schedule. And hence I want to reuse a given hardware registers for as many temporary registers as possible. This is how I do register minimization.

So, what are the properties of temporary registers? At any c step temporary register may appear only once on the LHS when it is written. And multiple times on RHS, when it is read. For example, this temporary register t 1 is written in time step one. At because it is at the output of operation one; subsequently this temporary register has been read in time step 2. So, it is read by this multiplication operation here, and this mult this addition operation here. This t 1 is read in both this operations and both these operations are in time step 2, right. If we take another example let us take t 2.

So, t 2 is written to in time step 1, and is read in time step 2 by this multiplication operator, and is read in time step 3 by this multiplication operator. So, a temporary register will occur at most once on the left hand side of the register transfer. And will appear and may appear multiple times on that RHS of a register transfer. And each register therefore, has a distinct life time interval from it is a first definition to it is last use. And hence this registers these temporary registers can have can be mapped to distinct intervals.

(Refer Slide Time: 06:48)



Register Minimization

Register Transfers at each c-step

1. $t1 \rightarrow x+y; t2 \rightarrow x \times z; t3 \rightarrow y+w$
2. $t5 \rightarrow t1 \times t2; t4 \rightarrow t1+t3$
3. $t6 \rightarrow t2 \times t4$
4. $t7 \rightarrow t5+t6$

Observation:
Registers with non-overlapping life times can be merged

1
2
3
4
5
t1
t2
t3
t4
t5
t6
t7

Life times of temporary registers

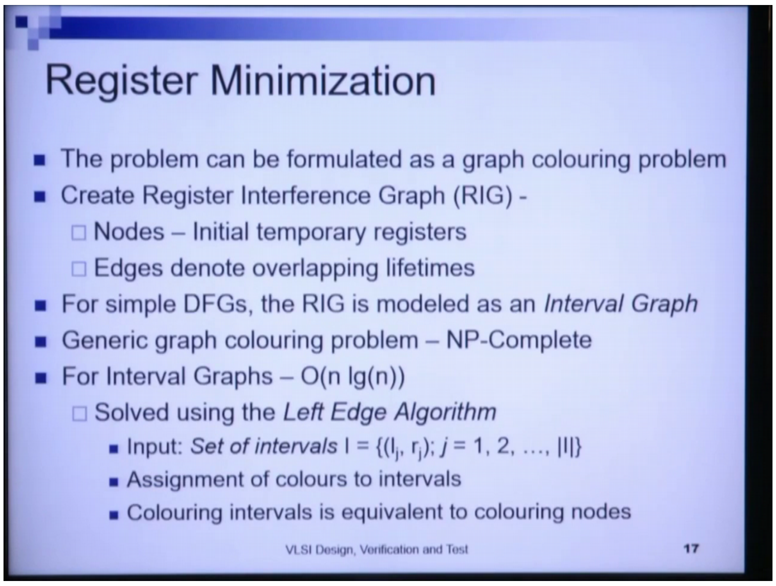
VLSI Design, Verification and Test 16

So, what do we infer from all these intervals? We infer that registers with non-overlapping lifetimes can be merged. So, temporary registers whose lifetimes are non-overlapping can be merged and allocated the same instance of a hardware register. For example, here t 1 and t 4 can be allocated to the same instance of register. And so, what is the, what do the algorithm basically look like here? We sort the intervals based on the left pages. And then we take one hardware register which is blank. And we take the first temporary register and put that in that hardware register.

Now say r 1 I have hardware register r 1 and I have put temporary register t 1 in to r 1 because, this has the earliest left page. Now I can put another register which is the earliest temporary register which is the earliest starting temporary register that I can put again in r one any register, whose left edge is higher than this right edge of t 1, right. So, therefore, t 4 is one search t 4 is the next search earliest starting temporary register which can be allocated to r 1. Similarly, I can also allocate t 6 to r 1 because t 4 and t 6 are non-overlapping the intervals are non-overlapping. So, I can put in to r 1 t 1 t 4 and t 6. I can even put t 7 in to r 1.

So, therefore, the same register r 1 can be used to whole t 1, t 4, t 6 and t 7; likewise, then I all these when I have already allocated. When I have already allocated a set of temporary registers to a given hardware registers. I now remove all these registers from consideration from the remaining. So, non-remaining temporary register can be allocated to the hardware register r 1. So, I put aside r one with it is allocation t 1, t 4, t 6 and t 7. Now, for the remaining register for now for the remaining temporary registers we try to allocate again the minimum number of hardware registers that that I need to allocate this temporary register in a very similar manner.

(Refer Slide Time: 09:32)



Register Minimization

- The problem can be formulated as a graph colouring problem
- Create Register Interference Graph (RIG) -
  - Nodes – Initial temporary registers
  - Edges denote overlapping lifetimes
- For simple DFGs, the RIG is modeled as an *Interval Graph*
- Generic graph colouring problem – NP-Complete
- For Interval Graphs – O(n lg(n))
  - Solved using the *Left Edge Algorithm*
    - Input: *Set of intervals* $I = \{(l_j, r_j); j = 1, 2, \ldots, |I|\}$
    - Assignment of colours to intervals
    - Colouring intervals is equivalent to colouring nodes

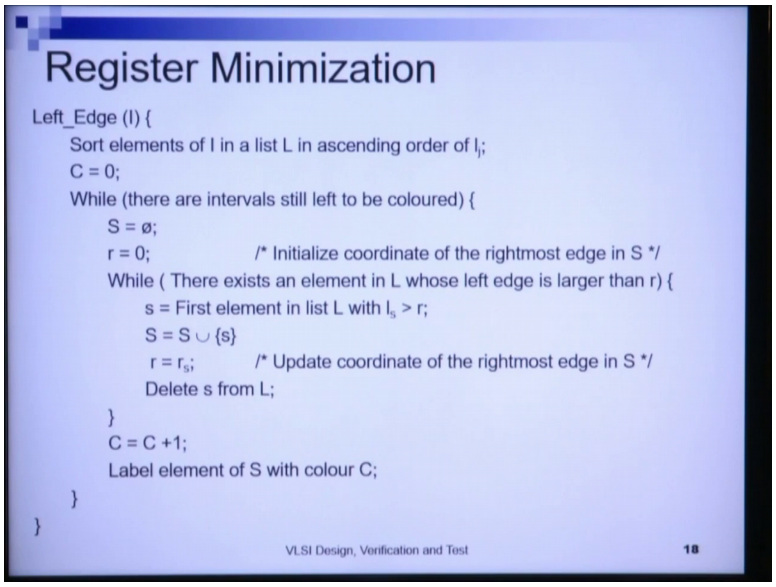
VLSI Design, Verification and Test 17

So, the algorithm that we have can be formulated as a graph coloring problem. So, this particular register conflict graphs. So, this is the register conflict graph, for the register conflict graph this register conflict graph also called register interference graph. Now, the register interference graph has nodes as the initial temporary registers. The edges denote overlapping lifetimes, right? For simple DFGs register interference graph is an interval graph, right similar to the fact that for simple DFGs the functional the behavioral operations in my operation constraints graph are also inter are also who also have a same simple continuous intervals.

And hence, their conflict graph can also be represented as an interval graph. Here again the interval graph or register conflict graph for a registers can also be similarly mapped as an interval graph. Now as we said the generic problem is NP complete; however, for interval graphs the optimal problem can be solved in polynomial time the optimal graph coloring can be sort of sort in polynomial time. And in fact, in n log n time and that also in n log n time, because prior to the algorithm I need to sort the left edges of all, the temporary registers in a list.

So, what are the left edges? Left edges are the start times the definition times the first definition times of the temporary registers. So, I need to keep a list which maintains the maintain all my temporary registers that have not yet been allocated in to a list sorted in terms of their start times their definition times or their left edges. So, in the interval graph I have a set of intervals I l g r j. So, this is the l j to r j is the lifetime for temporary register t j. Now we need to assign a distinct color to a temporary register. What does this assignment of color means? A distinct color if I assign to a temporary register mean that I have assigned a distinct hardware register to this temporary register. So, these list I of all temporary registers are taken as input by the left edge algorithm.

(Refer Slide Time: 12:17)



Register Minimization

```
Left_Edge (I) {
    Sort elements of I in a list L in ascending order of $l_i$;
    C = 0;
    While (there are intervals still left to be coloured) {
        S = ø;
        r = 0;              /* Initialize coordinate of the rightmost edge in S */
        While ( There exists an element in L whose left edge is larger than r) {
            s = First element in list L with $l_s > r$;
            S = S ∪ {s}
            r = $r_s$;      /* Update coordinate of the rightmost edge in S */
            Delete s from L;
        }
        C = C +1;
        Label element of S with colour C;
    }
}
```

VLSI Design, Verification and Test 18

Now, given this list of intervals it first sorts. These elements in I in a list l in ascending order of the left edges. First I assign 0 colors means c equals to 0. What does c means? C gives a labelling for colors. C equals to 1 is the distinct color. C equals to 2 is another distinct colors c equals to 3 is another distinct colors. So, if I allocate a color c equals to 1 to a set of temporary registers, then all those temporary registers will be will be allocated to the same hardware register whose color is labelled one, right.

So, why there are intervals still left be colored, will take a hardware register we will take a distinct color, and find out what are the maximum possible number of temporary registers that can be allocated to this hardware register. What is the maximum number of temporary registers that can be assigned as same color, ok. So, how do I do that? First S equals to phi; that means, S will contain S is a hardware register. I have initialized and currently it does not contain any temporary register in it, and hence S equals to phi.

S is the set of temporary register that can be allocated to single hardware registers in each given pass with device by look is a pass. Firstly, r equals to 0. The initial coordinate of the rightmost edge in S in the hardware register is set to 0. Why they are exist? An event in l whose left edge is larger than r; that means, that I want to find a left edge in the list l. And in the list l all my left edges are sorted are sorted in increasing order. So, first left edge beyond the value r in the in the register set in the in the hardware register defined by S currently if the register which I can allocate next to s.

So, small s is the first element in list l with ls greater than r, then s equals to s union small s; that means, I include this registers small s in the current set in the current hardware register being considered for allocation that is capital S. Then I change r equals to r s. So now, I change the right most edge in my hardware register r to the right edge of the current temporary register that has just allocated to s. Because the next temporary register that can be allocated to s will have a left edge which is greater than r s. Otherwise their intervals will be overlapping, and I will not be able to allocate in this same hardware register.

Now because s has small s has already been allocated over hardware register capital S I delete s from capital L which is the list of temporary register which have not yet been allocated. And then I assign the color then I assign c equals c plus 1 a labeling of color to capital S that is hardware register, ok. So, label element label all elements of s with color c, right. So, then all these temporary registers are allocated to the same hardware register and in color c. This loop will then continue until all temporary registers have been allocated hardware registers. So, like we can have we have applied this problem for register minimization.

I can likewise apply this problem for functional unit minimization as well. In that case what each type of operations that are schedulable by this same type of functional limit I will have to run this algorithm once to obtain the minimum number of functional unit instances that are required to allocate the behavioral operation of that type. So, this algorithm can be applied to both functional unit minimization as well as register minimization. With an understanding of the left edge algorithm, we come to the end of module one of lecture 7.