CS4272: HW SW Codesign

HW SW Partitioning

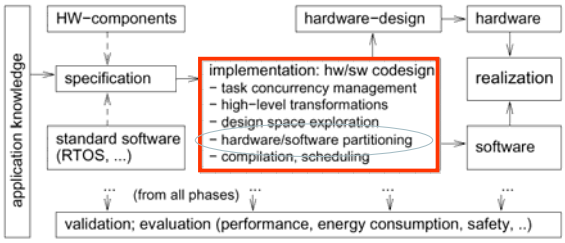
Abhik Roychoudhury
School of Computing
National University of Singapore

Modified and augmented from Peter Marwedel's lecture notes

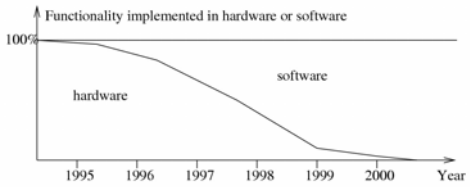
Reading

- Section 5.3 of textbook
  - Embedded System Design
  - Peter Marwedel
- Also must read
  - Hardware/software partitioning using Integer programming, by Ralf Niemann
  - URL available from CS4272 webpage.
  - This article has a much better explanation of the same material.

Hardware/Software Codesign



Hardware/software partitioning

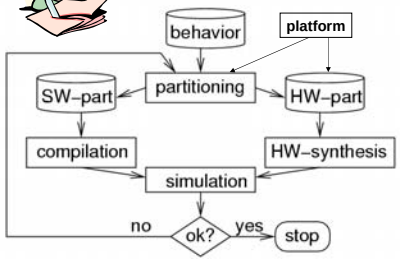


- No need to consider special purpose hardware in the long run?
- Correct for fixed functionality, but wrong in general, since "By the time MPEG-n can be implemented in software, MPEG-n+1 has been invented" [de Man]

☞Functionality to be implemented in software or in hardware?

Functionality to be implemented in software or in hardware?

- Decision based on hardware/ software partitioning, a special case of hardware/ software codesign.



Codesign Tool (COOL) as an example of HW/SW partitioning

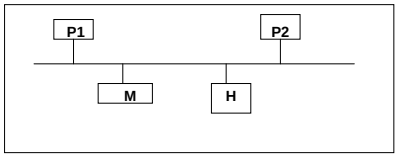
- Inputs to COOL:
  1. Target technology
  2. Design constraints
  3. Required behavior

Design constraints refer to constraints on performance, hardware area etc.

We need to clarify the other two terms.

Target Technology

- A graph of nodes --- Nodes denote
  - Hardware components or Processors
    - Memories also present, but
      - mapping of tasks to HW or Proc.
  - Edges denote interconnections --- often in the form of buses



So, Target Tech is

- Hardware Components
  - Possibly of different types
- Set of Processors
- External memory and buses between them.

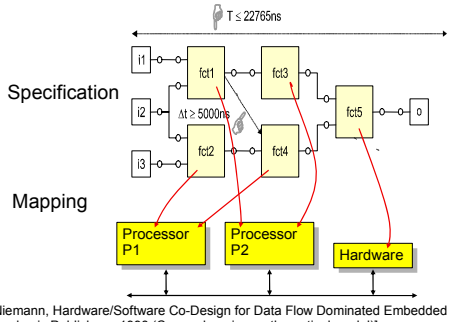
Behavior

- Hierarchical Task Graphs
- What is a task graph ?
  - Typically DAG of tasks
  - Nodes denotes specific tasks in the functionality of the system being designed
  - Edges can denote several things
    - Causal dependences, or in more details
    - Communication (with weightage of data being communicated)
    - There might exist causal dependence T1 → T2 without any data being communicated from T1 to T2
- Nodes of hierarchical task graphs can be task graphs

Why task graphs?

- Reasonable way to capture repetitive reactive behavior
  - Tasks produce output streams from input streams (from environment or other tasks).
- Task graphs thus represent a high-level behavioral specification of the system.
  - How each task is described depends on who is designing it (hardware designer, programmer)
  - Diff. from how each task will be implemented !

Approach

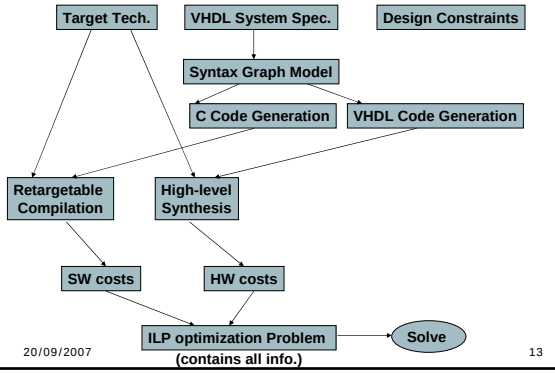


[Niemann, Hardware/Software Co-Design for Data Flow Dominated Embedded Systems, Kluwer Academic Publishers, 1998 (Comprehensive mathematical model)]

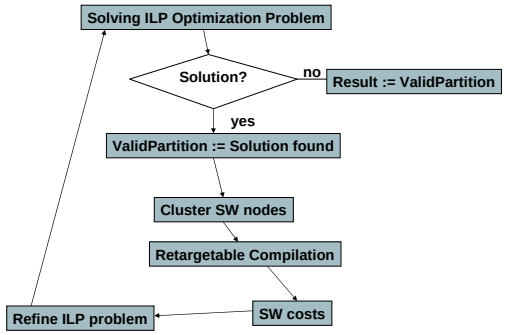
Task graph input

- Input to the partitioning method is a hierarchical task graph.
  - At the lowest level (leaves of the hierarchy), behavior of each node is specified say in VHDL
    - Alternately in C ?

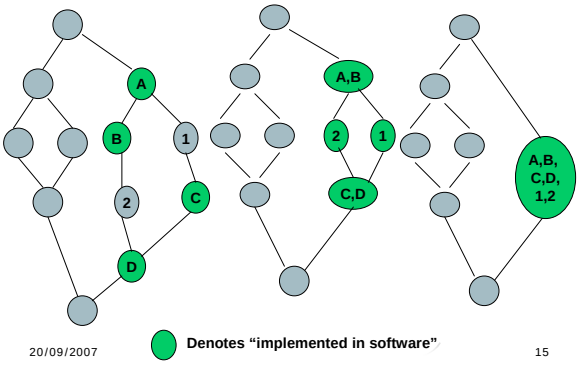
Schematic



Schematic (solving ILP)



Partition Refinement



COOL partitioning algorithm

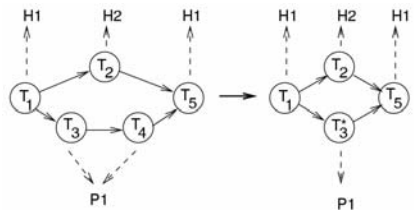
- **1. Translation of the behavior into an internal graph model**
- **2. Translation of the behavior of each node from VHDL into C (we assumed task description in VHDL, otherwise this step is not required).**
- **3. Compilation**
  - All C programs compiled for the target processor,
  - Computation of the resulting program size,
  - estimation of the resulting execution time (simulation input data might be required)
- **4. Synthesis of hardware components**:
  ∀ leaf node, application-specific hardware is synthesized.
  High-level synthesis sufficiently fast.

COOL partitioning algorithm

- **5. Flattening of the hierarchy**:
  Granularity used by the designer is maintained.
  Cost and performance information added to the nodes. Precise information required for partitioning is pre-computed
- **6. Generating and solving a mathematical model of the optimization problem**:
  Integer programming IP model for optimization.
  Optimal with respect to the cost function (approximates communication time)

COOL partitioning algorithm

- **7. Iterative improvements**:
  Adjacent nodes mapped to the same hardware component are now merged.



COOL partitioning algorithm

8. Interface synthesis:
After partitioning, the glue logic required for interfacing processors, application-specific hardware and memories is created.

We now describe step 6 (Integer Programming) in more details.

Integer programming models

- Ingredients:
- Cost function
- Constraints

} Involving linear expressions of integer variables from a set X

Cost function $C = \sum_{x_i \in X} a_i x_i \text{ with } a_i \in R, x_i \in \mathbb{N}$ (1)

Constraints: $\forall j \in J : \sum_{x_i \in X} b_{i,j} x_i \geq c_j \text{ with } b_{i,j}, c_j \in \mathbb{R}$ (2)

Def.: The problem of minimizing (1) subject to the constraints (2) is called an **integer programming (IP) problem**.

If all x_i are constrained to be either 0 or 1, the IP problem said to be a **0/1 integer programming problem**.

Example

$C = 5x_1 + 6x_2 + 4x_3$

$x_1 + x_2 + x_3 \geq 2$

$x_1, x_2, x_3 \in \{0,1\}$

x_1	x_2	x_3	C	
0	1	1	10	
1	0	1	9	← Optimal
1	1	0	11	
1	1	1	15	

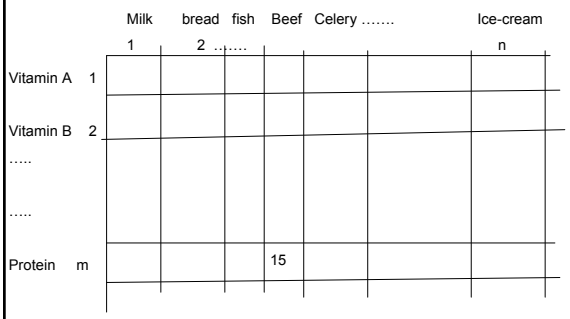
On integer programming

- Maximizing the cost function can be done by setting $C' = -C$
- Integer programming is NP-complete.
- In practice, running times can increase exponentially with the size of the problem, but problems of some thousands of variables can still be solved with commercial solvers, depending on the size and structure of the problem.
- IP models can be a good starting point for modeling, even if in the end heuristics have to be used to solve them.

Digress: Linear Programming

- Example: Grocery Shopping
  - m varieties of nutrients (vitamins, protein, ..)
    - need b_1 units of Nut_1, b_2 units of Nut_2,..., b_m units of Nut_m.
  - Can buy n types of food (milk, bread, beef,..)
  - Each unit of food contains a certain number of units of each type of nutrients.
  - $a_{i,j}$ represents the number of units of the ith type nutrient contained in one unit of food of the jth type.

Digress: Linear Programming



Digress: Linear Programming

- Suppose you buy x_1 units of food type 1 and x_2 units of food type 2,... and x_n units of food type n. Then for nutrition type of type i it must be the case :

  $a_{i,1} . x_1 + a_{i,2} . x_2 + ... + a_{i,n} . x_n \geq b_i$

Digress: Linear Programming

$a_{1,1} . x_1 + a_{1,2} . x_2 + ... + a_{1,n} . x_n \geq b_1$

$a_{2,1} . x_1 + a_{2,2} . x_2 + ... + a_{2,n} . x_n \geq b_2$

$a_{i,1} . x_1 + a_{i,2} . x_2 + ... + a_{i,n} . x_n \geq b_i$

$a_{m,1,} . x_1 + a_{m,2} . x_2 + ... + a_{m,n} . x_n \geq b_m$

Digress: Linear programming

- **A X** $\geq$ **b**
- **A** - m $\times$ n matrix
- **X** - n $\times$ 1 column vector of unknowns.
- **b** - m $\times$ 1 column vector of constants.
- Additional constraints
  - $x_j \geq 0$ (j = 1, 2, ...n)
- Cost function:
  - $z = c_1 . x_1 + c_2 . x_2 + ...+ c_n . x_n$
  - c_i - the cost of one unit of food type i.

Digress: Linear Programming

- The LP Problem.
  - Find $(x_1, x_2, ..., x_n)$ such that:
    - All the constraints are satisfied.
    - The cost function is minimized.
      - If $(y_1, y_2, ..., y_n)$ also satisfies all the constraints then $z' \geq z$ where:
      - $z = c_1 . x_1 + c_2 . x_2 + ...+ c_n . X_n$
      - $z' = c_1 . y_1 + c_2 . y_2 + ...+ c_n . y_n$

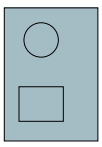
Integer Linear Programming

- Demand in addition:
  - Each x_i should be an integer.
- Solving an ILP problem usually boils down to solving a series of LP problems.
- The General Idea in solving an LP:
  - Feasible solution is a solution that satisfies all the constraints.
  - The set of feasible solutions (for sensible LP problems!) is a convex polyhedron.
  - One of the corner points of the polyhedron is the optimal solution.

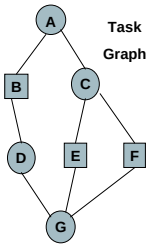
Mixed Integer Linear Programming Problem

- Demand the integer-value constraint only for a subset of the variables.
- In principle, LP problems can be solved in polynomial time.
- But ILP problems have only exponential time algorithms at present.
  - NP-complete
- Role of ILP in solving co-design problems
  - ILP based resource aware compilation – Palsberg and Naik
  - http://www.cs.ucla.edu/~palsberg/paper/mpsoc-chapter03.pdf

The Partitioning Problem

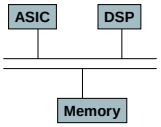


Set of entities or tasks



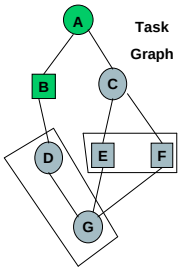
Task Graph

Target Tech.



Partitioning Problem: Map {A,B,C,D.E,F,G} to {ASIC, DSP}.

Possible solution



Task Graph

Shared implementation

Green color: executing in DSP

IP model for partitioning

- **Notation:**
- Index set *I* denotes task graph nodes.
- Index set *L* denotes task graph node **types**
e.g. square root, DCT or FFT
- Index set *KH* denotes hardware component **types**.
e.g. hardware components for the DCT or the FFT.
- Index set *J* of hardware component instances
- Index set *KP* denotes processors.
All processors are assumed to be of the same type

IP model for partitioning

- $X_{i,k}$: =1 if node v_i is mapped to hardware component type $k \in KH$ and 0 otherwise.
- $Y_{i,k}$: =1 if node v_i is mapped to processor $k \in KP$ and 0 otherwise.
- $NY_{\ell,k}$ =1 if at least one node of type ℓ is mapped to processor $k \in KP$ and 0 otherwise.
- T is a mapping from task graph nodes to their types:
$T: I \rightarrow L$
- The cost function accumulates the costs:
C = cost(processors) + cost(memories) + cost(application specific hardware)

Constraints

- Operation assignment constraints

$$\forall i \in I : \sum_{k \in KH} X_{i,k} + \sum_{k \in KP} Y_{i,k} = 1$$

All task graph nodes have to be mapped either in software or in hardware.

Variables are assumed to be integers.

Additional constraints to guarantee they are either 0 or 1:

$$\forall i \in I : \forall k \in KH : X_{i,k} \leq 1$$

$$\forall i \in I : \forall k \in KP : Y_{i,k} \leq 1$$

Operation assignment constraints (2)

$\forall \ell \in L, \forall i: T(v_i) = \ell, \forall k \in KP: NY_{\ell,k} \geq Y_{i,k}$

- For all types ℓ of operations and for all nodes i of this type: if i is mapped to some processor k, then that processor must implement the functionality of ℓ.

- Decision variables must also be 0/1 variables:
$\forall \ell \in L, \forall k \in KP: NY_{\ell,k} \leq 1$.

Resource & design constraints

- $\forall\ k \in KH$, the cost (area) used for components of that type is calculated as the sum of the costs of the components of that type. This cost should not exceed its maximum.
- $\forall\ k \in KP$, the cost for associated data storage area should not exceed its maximum.
- $\forall\ k \in KP$ the cost for storing instructions should not exceed its maximum.
- The total cost ($\Sigma_{k \in KH}$) of HW components should not exceed its maximum
- The total cost of data memories ($\Sigma_{k \in KP}$) should not exceed its maximum
- The total cost instruction memories ($\Sigma_{k \in KP}$) should not exceed its maximum

Timing constraints

Timing constraints
These constraints can be used to guarantee that certain time constraints are met.

Execution time of a node in the task graph is variable (implemented in hardware or software)

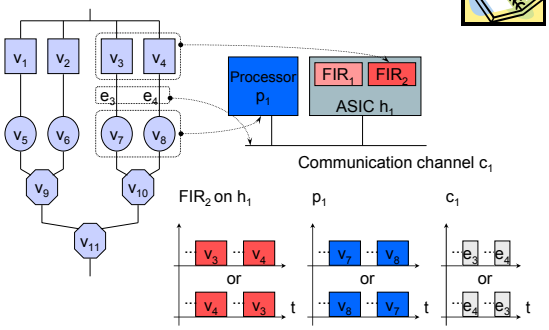
This defines execution time of node as a linear expression on our decision variables.

Using these execution times, we can define start and end times of each node.

The end time of the sink node in the task graph should be less than pre-defined constant

--- overall timing constraint on the design.

Scheduling



Scheduling / precedence constraints

- For all nodes v_{i1} and v_{i2} that are potentially mapped to the same processor or hardware component instance, introduce a binary decision variable $b_{i1,i2,k}$ with $b_{i1,i2,k}=1$ if v_{i1} is executed before v_{i2} *in component k* and $= 0$ otherwise.
  Define constraints of the type
  (end-time of v_{i2}) $\leq$ (start time of v_{i1}) if $b_{i1,i2,k}=0$
- Ensure that the schedule for executing operations is consistent with the precedence constraints in the task graph.

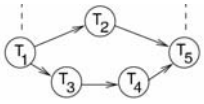
Scheduling Constraints

- $b_{i1,i2,k} + y_{i1,k} \geq 1$
- $b_{i1,i2,k} + y_{i2,k} \geq 1$
- $b_{i1,i2,k} + b_{i2,i1,k} \geq 1$
- $b_{i1,i2,k} + b_{i2,i1,k} + y_{i1,k} + y_{i2,k} \leq 3$

- $Time_{i1}^{start} \geq Time_{i2}^{end} - (\text{Large Const}) * b_{i1,i2,k}$

- $Time_{i2}^{start} \geq Time_{i1}^{end} - (\text{Large Const}) * b_{i2,i1,k}$

Example



- HW types H1, H2 and H3 with costs of 20, 25, and 30.
- Processors of type P.
- Tasks T1 to T5.
- Execution times:

T	H1	H2	H3	P
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

Operation assignment constraints (1)

T	H1	H2	H3	P
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

$$\forall i \in I: \sum_{k \in KH} X_{i,k} + \sum_{k \in KP} Y_{i,k} = 1$$

$X_{1,1}+Y_{1,1}=1$ (task 1 mapped to H1 or to P)
$X_{2,2}+Y_{2,1}=1$
$X_{3,3}+Y_{3,1}=1$
$X_{4,3}+Y_{4,1}=1$
$X_{5,1}+Y_{5,1}=1$

Operation assignment constraints (2)

- Assume types of tasks are ℓ =1, 2, 3, 3, and 1.

$\forall \ell \in L, \forall i: T(v_i) = \ell \; \forall k \in KP: NY_{\ell,k} \geq Y_{i,k}$

$NY_{1,1} \geq Y_{1,1}$
$NY_{2,1} \geq Y_{2,1}$
$NY_{3,1} \geq Y_{3,1}$
$NY_{3,1} \geq Y_{4,1}$
$NY_{1,1} \geq Y_{5,1}$

Functionality 3 to be implemented on processor if node 4 is mapped to it.

Other equations

- Time constraints leading to: Application specific hardware required for time constraints under 100 time units.

T	H1	H2	H3	P
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

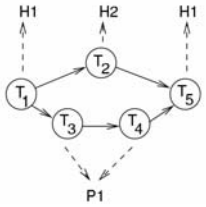
Cost function:
C=20 #(H1) + 25 #(H2) + 30 # (H3) + cost(processor) + cost(memory)

Result

- For a time constraint of 100 time units and cost(P)<cost(H3):

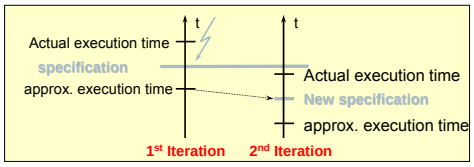
T	H1	H2	H3	P
1	20			100
2		20		100
3			12	10
4			12	10
5	20			100

Solution (educated guessing) :
T1 → H1
T2 → H2
T3 → P
T4 → P
T5 → H1



Separation of scheduling and partitioning

- Combined scheduling/partitioning very complex;
  ☞ Heuristic: Compute approx. time values.
- Perform partitioning for approx. time values.
- Perform final scheduling using the above partition.
- If final schedule does not meet time constraint, go to 1 using a reduced overall timing constraint.



How to get approx time?

- Compute start and end times of each node without the scheduling constraints.
  - Only basic constraints based on topological ordering in the task graph.
  - $Time_i^{start} \geq \Sigma_{predecessors\ j}\ Y_{j,k}$ * (sw time of j on k)
    - Similarly for hardware
  - $Time_i^{start} \geq Time_j^{end} +$
  - $\Sigma_{j' \in path(j,i)}\ Y_{j',k}$ * (sw time of j' on k)

  where j is the dominator of i in task graph.
  (similar constraints for hardware costs).

Compute partitioning with these time values.
Then solve the scheduling using the given partition.

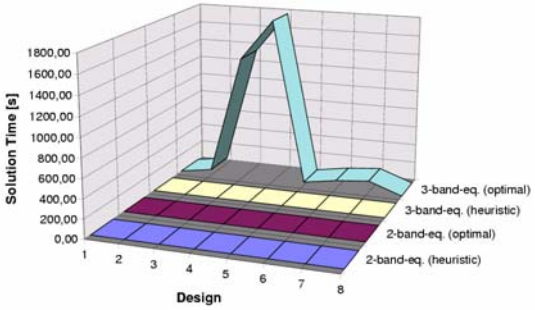
Application example

- Audio lab (mixer, fader, echo, equalizer, balance units); slow SPARC processor
- 1µ ASIC library
- Allowable delay of 22.675 µs (~ 44.1 kHz)

SPARC processor | ASIC (Compass, 1 µ) | External memory

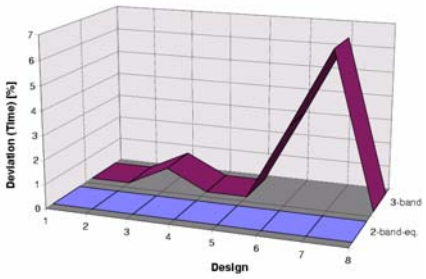
Outdated technology; just a proof of concept.

Running time for COOL optimization



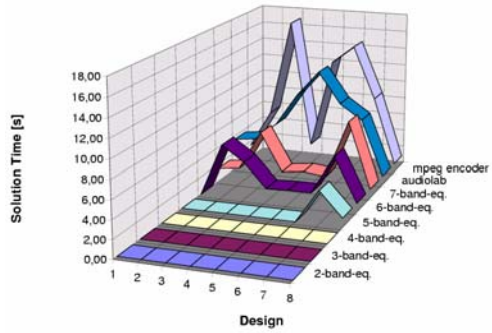
☞ Only simple models can be solved optimally.

Deviation from optimal design

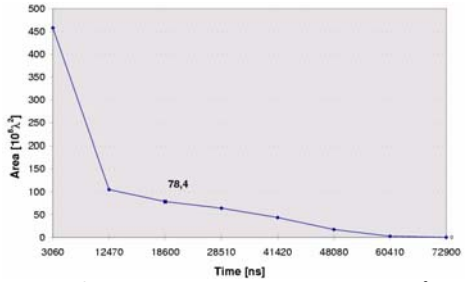


☞ Hardly any loss in design quality.

Running time for heuristic



Design space for audio lab

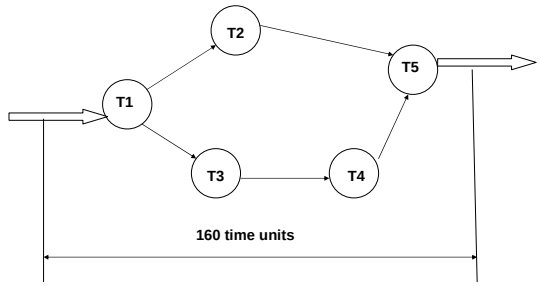


Everything in software: 72.9 µs, 0 λ^2
Everything in hardware: 3.06 µs, 457.9x10^6 λ^2
Lowest cost for given sample rate: 18.6 µs, 78.4x10^6 λ^2,

Final remarks

- COOL approach:
  - shows that formal model of hardware/SW codesign is beneficial; IP modeling can lead to useful implementation even if optimal result is available only for small designs.
- Other approaches for HW/SW partitioning:
  - starting with everything mapped to hardware; gradually moving to software as long as timing constraint is met.
  - starting with everything mapped to software; gradually moving to hardware until timing constraint is met.
  - Simple approaches like Binary search.

Exercise



Exercise

- At most 2 nodes can be implemented in HW ASIC. Each ASIC costs 20 units and each software implementation costs 5 units.
- Each task's HW execution is 8 time units and the SW execution is 60 time units.
- Total execution time should be less than 160 time units.
- Perform HW-SW partitioning for this task graph.

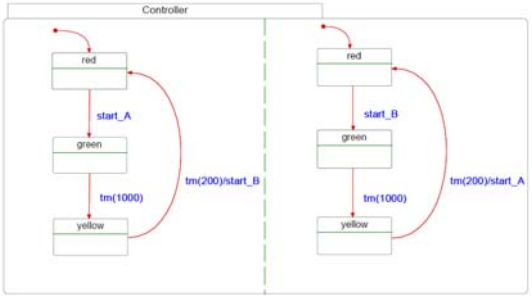
Exercises

A. Consider a traffic light controller with two different lights each of which must be red when the other is not red. They both start in the red state. When one of them receives a start event, it performs a cycle going from red to green to yellow and back to red. When either light reaches the red state, it tells the other to perform a cycle.

1. **Draw the above as a statechart.**
2. **Identify what features of statechart you found most useful ?**

B. Event broadcasting allows output of a transition to serve as triggers of transitions in orthogonal components of a system. Can such broadcast go on in an infinite loop ? Construct a small example statechart to show that this is possible.

Exercise A



tm(n): a timeout transition triggered after a specified amount of time (n) has passed.

Exercise B