MECHTRON 3TB4
HW/SW Codesign

Motivating Example

Consider a Digital TV box. The main functions are:

- receiving encrypted video data
- decompression of received video, compression of received video (for storage)
- image filtering (quality improvement)

There are most likely a number of other functions required:

- channel guide
- potential multimedia uses (store photos, music, etc.)
- downloadable apps

Implementation Choices - I

Which processing components should you consider?

- general purpose processors
- specialized processors (DSP, microcontroller, etc.)
- FPGA
- ASIC

Implementation Choices - II

How should you divide the processing among resources?

- everything on one resource (for example, all in software for a single CPU)
- build a system from dedicated processors
- build a system-on-chip (SOC)
- mix and match

Implementation Choices - III

How to interface the components?

- point-to-point communication links
- bus architectures

Impact of Choices

- performance of different target architectures varies widely
- one example is implementation of AES-128 encryption algorithm
- here are some representative figures
  1. general purpose CPU, 7 KB of code, about 100 clock cycles per bit
  2. hardware implementation 1: approx. 3500 gates, 10 clock cycles per bit
  3. hardware implementation 2: approx. 5500 gates, 2 bits per clock cycle
- tradeoff of performance vs. complexity

Impact of Choices (cont'd)

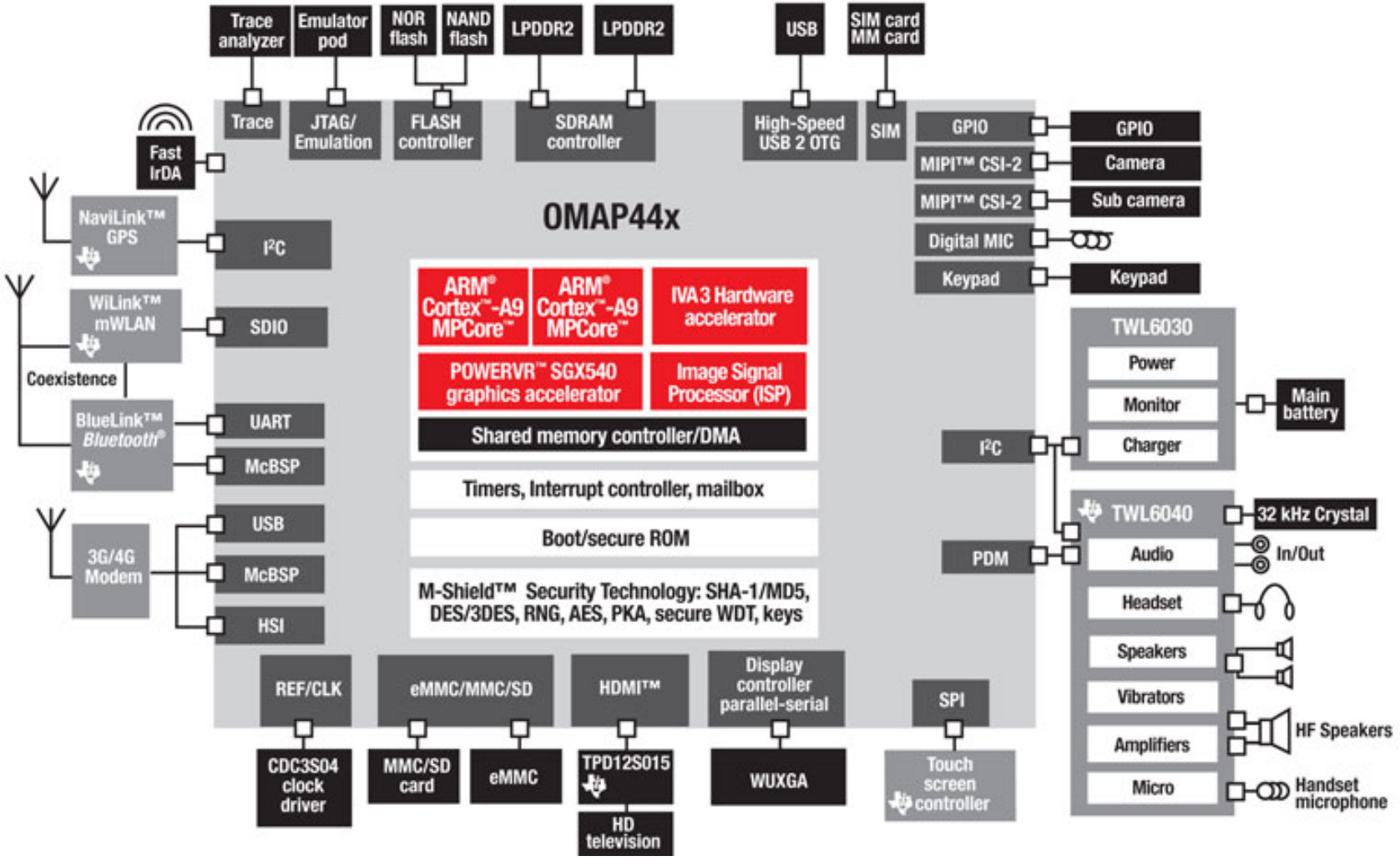
Energy efficiency is even a bigger issue:

1. C code on general purpose CPU, approximately 10^{-3} Gb/J
2. FPGA, approximately 1 Gb/J
3. ASIC, approximately 10 Gb/J

Mix of Architectures

1. constraints usually mean that a single choice for all is not suitable
   - requirements on performance, power, cost
   - cost may be impacted by reuse (HW or SW)
   - may be able to amortize development costs over a variety of products
2. most applications use a mixture of HW/SW technologies
   - programmable processors
   - fixed co-processors/accelerators
   - reconfigurable accelerators
3. functional partition depending on requirements
   - fixed functions that are performance or power constrained call for specified accelerators
   - variable or noncritical functions on programmable processors

System-on-Chip



(Image from Texas Instruments)

ASIC Examples

- bitcoin mining (Antminer S19j)
  - high throughput
  - high cost, high power consumption (more than 3 kWatt), large
- LED integrated light source (WS2812): cheap and easy lighting solutions
- hearing aids: voltage regulation for AgZn batteries, power consumption, specialized signal processing
- many companies working in this design support space
- fully custom, semi-custom

Key Questions

1. hardware platform
   - what processing elements are available?
   - how to choose those that satisfy performance, cost, power budget, etc.?
   - how to communicate between components?
2. application
   - how to decompose the application?
   - where to execute each part?
3. tools
   - are there tools available to support the design process?

Approaches

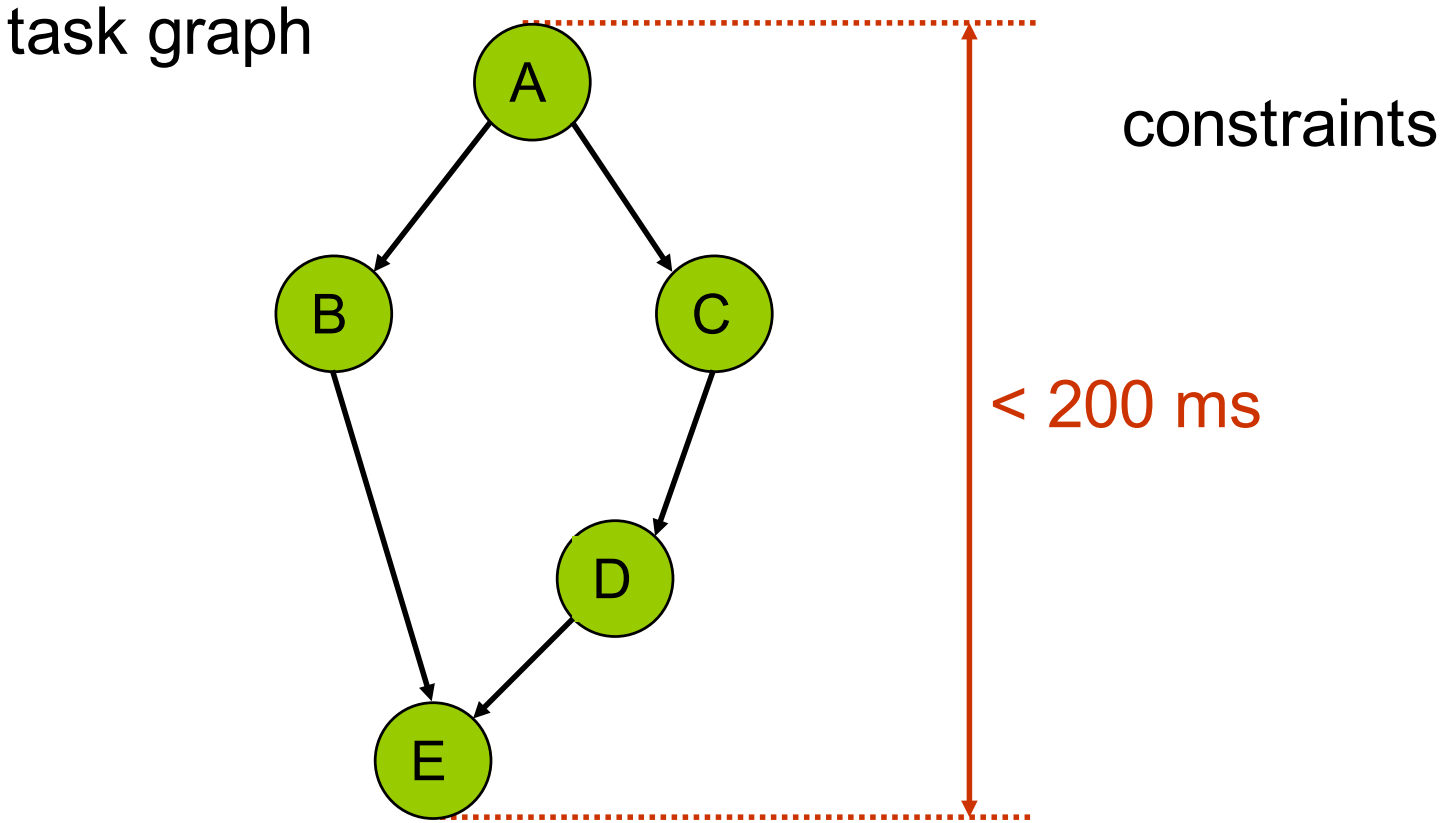
1. treat design of HW/SW system as a formal optimization problem
2. use a more incremental approach

Synthesis Example

- transform behavioural description into a structural description
- synthesis essentially has three tasks
  1. **allocation**: select components
  2. **binding**: assign functions to components
  3. **scheduling**: determine execution order

Synthesis Example (cont'd)

- behaviour specified by the following task graph



- constraint is that completion time must be less than some required time

Synthesis Example: Allocation

We have three possible components:

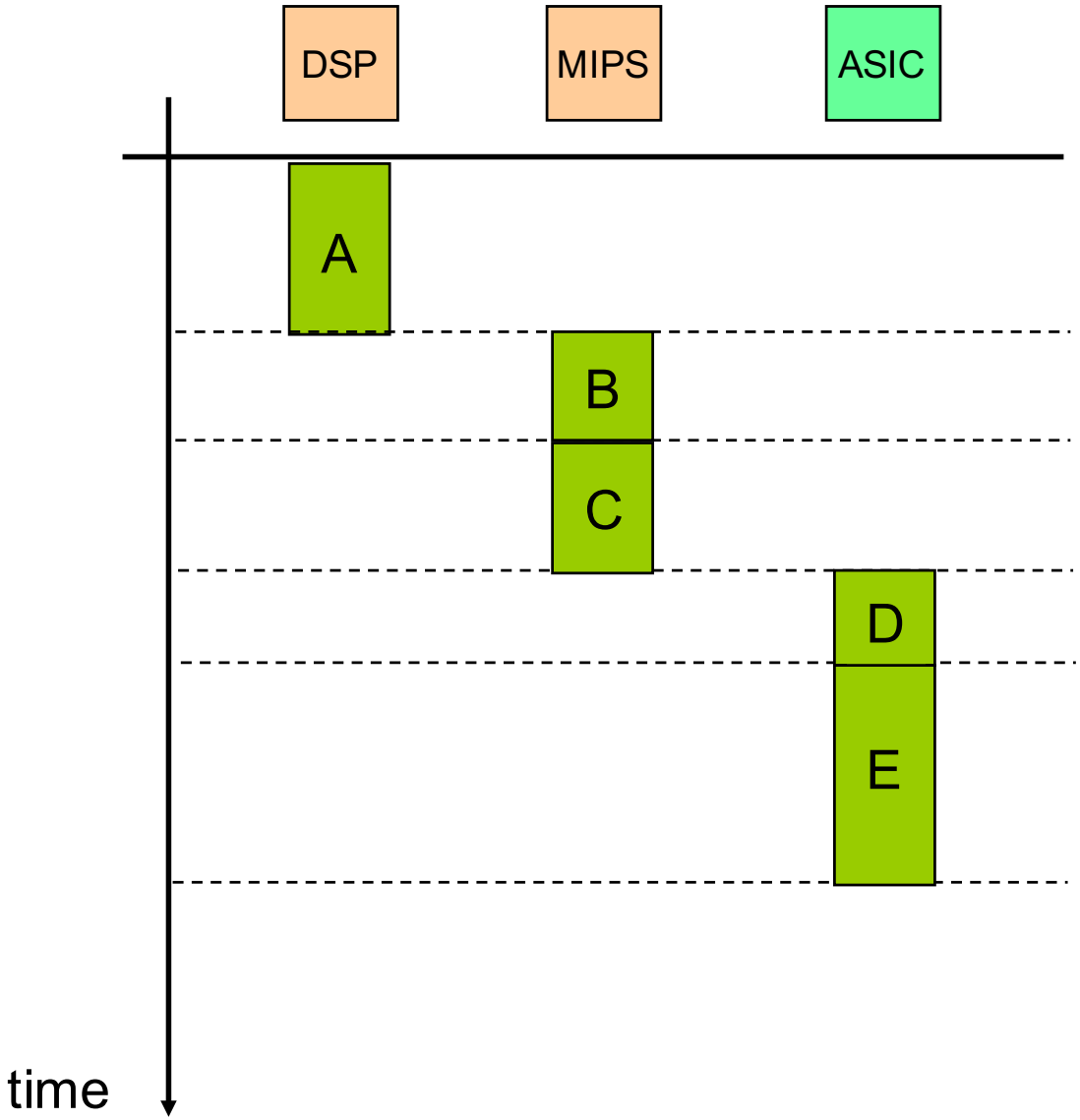
1. MIPS (general purpose processor)
2. DSP
3. ASIC

Synthesis Example: Binding

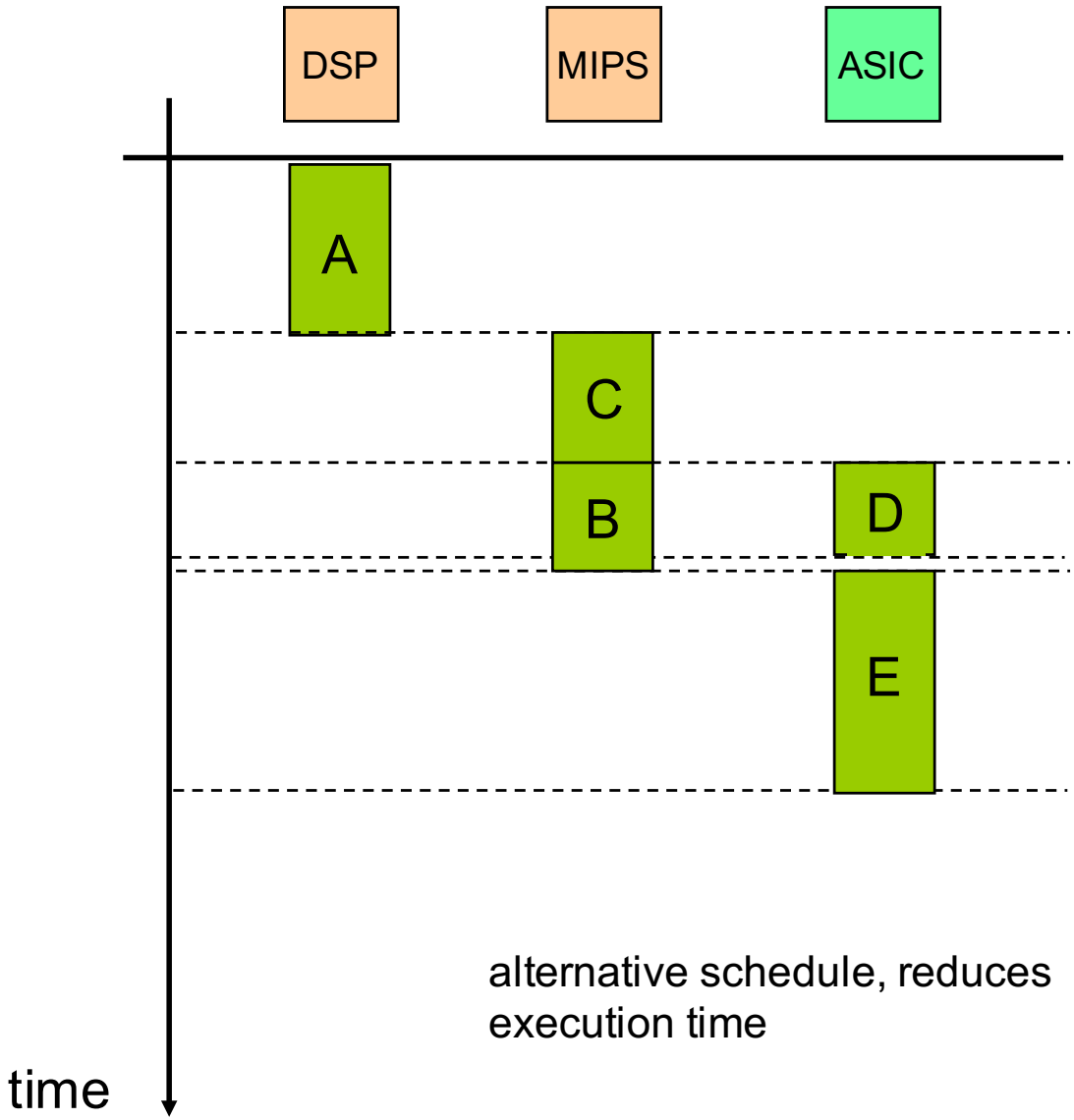
Suppose that we make the following choices:

1. Task A: DSP
2. Tasks B and C: MIPS
3. Tasks D and E: ASIC

Synthesis Example: Scheduling



Synthesis Example: Scheduling



HW/SW Partitioning – Formal Optimization

Overview

- formulate binding/scheduling problem as a formal optimization problem
- two examples to give the idea
- some additional references will be posted

First Example

- an application consists of four tasks, A, B, C, D that can all be done in parallel
- define t_A, t_B, t_C, t_D to be the execution times for the tasks
- similarly, define e_A, e_B, e_C, e_D to be the energy consumption for each of the tasks
- the available components are two processors, P_1 and P_2
- goal is to minimize energy consumption subject to a constraint on the latency for the application

	t_A	t_B	t_C	t_D
P_1	5	15	10	30
P_2	10	20	10	10

	e_A	e_B	e_C	e_D
P_1	10	6	3	1
P_2	3	8	3	3

Partitioning

- example of a partitioning problem, need to partition tasks between the two processors
- introduce variables that encode the partition, for example $x_{A,1} = 1$ if task A is assigned to P_1 and 0 if it is not. Similarly, $x_{A,2} = 1$ if task A is assigned to P_2 and 0 if it is not
- similar binary variables for other tasks

Constraints - I

First, each task must be assigned to a processor:

$$\begin{aligned} x_{A,1} + x_{A,2} &\geq 1 \\ x_{B,1} + x_{B,2} &\geq 1 \\ x_{C,1} + x_{C,2} &\geq 1 \\ x_{D,1} + x_{D,2} &\geq 1 \end{aligned}$$

Constraints - II

Latency constraint. Suppose that we want that the application latency is at most 30. This means the total processing time of **each** processor must be at most 30.

$$\begin{aligned} 5x_{A,1} + 15x_{B,1} + 10x_{C,1} + 30x_{D,1} &\leq 30 \\ 10x_{A,2} + 20x_{B,2} + 10x_{C,2} + 10x_{D,2} &\leq 30 \end{aligned}$$

Objective Function

This is what we would like to minimize, in this case the total energy consumption:

$$10x_{A,1} + 6x_{B,1} + 3x_{C,1} + x_{D,1} + 3x_{A,2} + 8x_{B,2} + 3x_{C,2} + 3x_{D,2}$$

Overall problem

minimize

$$10x_{A,1} + 6x_{B,1} + 3x_{C,1} + x_{D,1} + 3x_{A,2} + 8x_{B,2} + 3x_{C,2} + 3x_{D,2}$$

subject to

$$\begin{aligned}
x_{A,1} + x_{A,2} &\geq 1 \\
x_{B,1} + x_{B,2} &\geq 1 \\
x_{C,1} + x_{C,2} &\geq 1 \\
x_{D,1} + x_{D,2} &\geq 1 \\
5x_{A,1} + 15x_{B,1} + 10x_{C,1} + 30x_{D,1} &\leq 30 \\
10x_{A,2} + 20x_{B,2} + 10x_{C,2} + 10x_{D,2} &\leq 30 \\
x_{j,k} &\in \{0, 1\}
\end{aligned}$$

Integer Linear Programming

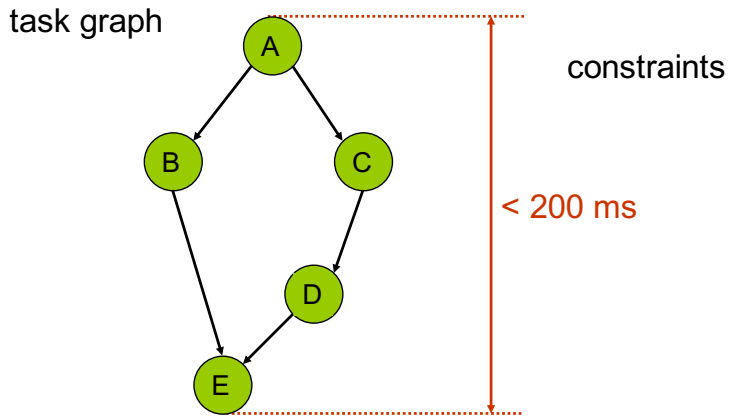
- this is an example of an *integer linear programming problem*, or *integer linear program*
- variables are integer valued (in this case binary) and constraints are linear
- known to be NP-complete (efficient algorithms do not exist for large problems)
- small instances can still be solved
- `intlinprog` in Matlab

Solution

- using Matlab, I got a solution of (only elements that are one are listed): $x_{B,1} = 1$, $x_{A,2} = 1$, $x_{C,2} = 1$, $x_{D,2} = 1$, with a total energy consumption of 15
- one attractive aspect of this approach is that one can formally examine trade-offs
- for example, what if we relax the latency constraint to 45
- solution of $x_{B,1} = 1$, $x_{D,1} = 1$, $x_{A,2} = 1$, $x_{C,2} = 1$ with a total energy consumption of 13

One More Example

Let's use the task graph from earlier:



- components available: HW types $H1$, $H2$, $H3$ with costs of 20, 25, 30, processor of type P, cost 10
- wish to minimize system cost subject to latency constraint

Execution Times

	$H1$	$H2$	$H3$	P
A	20			100
B		20		100
C			12	10
D			12	10
E	20			100

Constraints - I

As before, each task must be assigned to a processor. Here, I have numbered the components: 1 ($H1$), 2 ($H2$), 3 ($H3$) and 4 (P).

$$
\begin{aligned}
x_{A,1} + x_{A,4} &\geq 1 \\
x_{B,2} + x_{B,4} &\geq 1 \\
x_{C,3} + x_{C,4} &\geq 1 \\
x_{D,3} + x_{D,4} &\geq 1 \\
x_{E,1} + x_{E,4} &\geq 1
\end{aligned}
$$

Constraints - II

- As before, also need latency constraints.
- Here, we will use an approximation, treating the two paths as independent (they are not).
- I will write these first for a constraint of 100 (as opposed to the 200 in the task graph).

$$
\begin{aligned}
20x_{A,1} + 100x_{A,4} + 20x_{B,2} + 100x_{B,4} + 20x_{E,1} + 100x_{E,4} &\leq 100 \\
20x_{A,1} + 100x_{A,4} + 12x_{C,3} + 10x_{C,4} + 12x_{D,3} + 10x_{D,4} + 20x_{E,1} + 100x_{E,4} &\leq 100
\end{aligned}
$$

Constraints - III

For $H1$, $H3$ and P we also need a variable that tells us if the component is used or not ($x_{B,2}$ does this for $H2$).

$$
\begin{aligned}
x_1 &\geq x_{A,1} \\
x_1 &\geq x_{E,1} \\
x_3 &\geq x_{C,3} \\
x_3 &\geq x_{D,3} \\
x_4 &\geq x_{A,4} \\
x_4 &\geq x_{B,4} \\
x_4 &\geq x_{C,4} \\
x_4 &\geq x_{D,4} \\
x_4 &\geq x_{E,4}
\end{aligned}
$$

Objective function

Here, we would like to minimize

$$20x_1 + 25x_{B,2} + 30x_3 + 10x_4$$

Solution

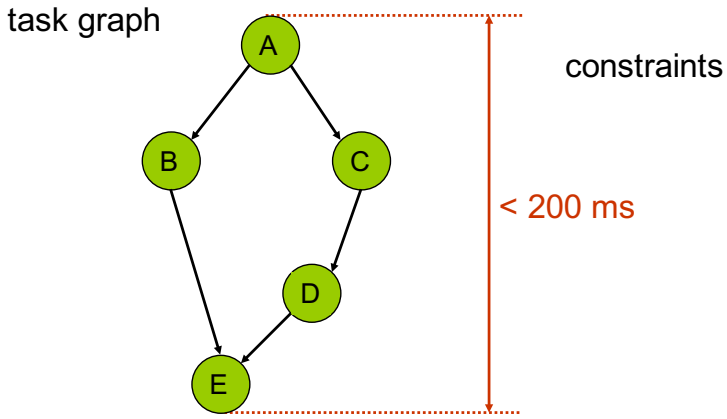
- nonzero assignments are $x_{A,1} = 1$, $x_{B,2} = 1$, $x_{C,4} = 1$, $x_{D,4} = 1$, $x_{E,1} = 1$
- suppose that we relax the latency constraint to 200
- nonzero assignments are $x_{A,1} = 1$, $x_{B,4} = 1$, $x_{C,4} = 1$, $x_{D,4} = 1$, $x_{E,1} = 1$
- complete problem statement plus how I solved it in Matlab in accompanying notes (for latency constraint of 200)

Remarks

- there are difficulties in scaling this technique
- generating constraints can be difficult, in particular latency constraints are not so easy to write when the task graph is complicated
- many heuristic approaches have been developed
- does show value of formalization of a problem

HW/SW Partitioning – Incremental Approach

Revisiting previous example



- components available: HW types $H1$, $H2$, $H3$ with costs of 20, 25, 30, processor of type P, cost 10
- wish to minimize system cost subject to latency constraint

Execution Times

	$H1$	$H2$	$H3$	P
A	20			100
B		20		100
C			12	10
D			12	10
E	20			100

Incremental Solution

- start with minimum cost solution: all of the processing is done on P, with resulting total cost of 10
- at this point, adding task A on $H1$ would cause greatest reduction in latency at lowest additional cost, reduce latency by 80, increase cost by 20
- this is not a unique choice (could add task E instead and get the same results)
- next, add task E on $H1$, reduce latency by 80, no additional cost
- can continue in this manner until an acceptable solution is found (may not find best possible)

Remarks

- incremental approach is straightforward in this case, as all required parameters known
- in general, all values are not known in advance, plus tradeoffs are difficult to quantify in a single cost function
- however, incremental approach is still quite useful, if one can at least identify where improvement is required and (rough) estimates can be made for the degree of improvement that is possible

Digital Camera Example

- Chapter 7 of "Embedded System Design: A Unified Hardware/Software Introduction" by Vahid and Givargis: some numbers out of date, but concepts still very much valid
- highlight some issues here, but will not go into same depth as book
- I will post a slide deck from the author of the book, these are pretty comprehensive

High Level Description

- captures and stores images in digital format
- key issue: compression of images used to increase the number of images that can be stored in memory (and decrease transmission time of an image)

Raw Data

- CCD captures image
- assume black and white image, each pixel of CCD is an 8-bit value (0 to 255) representing exposure of cell to light
- assume that we have a 64 pixel by 64 pixel image

Pre-processing: Zero-Bias Error

- adjust for bias due to manufacturing errors
- done by covering cells at end of row with black paint (should read zero)
- bias tends to be constant across rows

JPEG compression

- JPEG standard has a number of modes
- here, the following steps are performed on a block of data (size 8 pixels by 8 pixels)
  1. Discrete Cosine Transform (good for images)
  2. Quantization (reduction of size: 8 bits to k bits)
  3. Huffman encoding (additional reduction)
- will not go into details here

Design Considerations

1. latency: 1 second to process image (constraint)
2. size: make as small as possible, but must be below a certain size (constraint plus optimization)
3. power: constraint due to thermal considerations
4. energy: reducing power or latency reduces energy consumption, would like to optimize (make battery last as long as possible)
5. time to market: affects potential profits
6. profit: optimize

Initial Software Prototype

- starting point is (cheap) general-purpose processor connected to memory
- automatically satisfies power, size and time-to-market constraints

Implementation 1: Microcontroller

- half of latency budget taken by loading and performing zero-bias correction for CCD data
- no hope of meeting latency constraint as compression has not even been performed

Implementation 2: Microcontroller plus CCDPP accelerator

- accelerator to load data and perform zero-bias correction (read entire row of CCD at a time and perform bias correction in parallel)
- requires interfaces to be designed, as described in detail in slides (bus protocol as discussed earlier in the course)
- helpful that HDL implementation of microcontroller was available

Implementation 2: Results

- all is good, except still exceed latency requirement by a factor of 9

Implementation 3: Fixed Point Calculations

- at this point, could design accelerator for compression stage
- an intermediate possibility is to move calculations from floating point to fixed point (in particular for DCT)
- floating point is much more expensive than fixed point (can be exacerbated for processors that emulate floating point)
- do all calculations in fixed point (but still in software)
- considerations similar to what you did in the lab for the FIR filter implementation

Implementation 4: Compression Acceleration

- Implementation 3 not good enough (latency of 1.5 seconds)
- design compression accelerator (it is a good candidate)
- results in performance well under constraint

Final Remarks

- perhaps Implementation 3 is good enough?
- additional development time may be problematic