COE718: Embedded Systems Design

Lecture 12:

HW/SW Co-Design & Accelerator based Embedded System Design

Anita Tino

Processors – Adv / Limitations

Processors – Adv / Limitations

Advantages

- Easy to program
- Portability support (ISA, compilers etc are well-known and optimized)
- OS support
- Able to process any application

Processors – Adv / Limitations

Advantages

- Easy to program
- Portability support (ISA, compilers etc are well-known and optimized)
- OS support
- Able to process any application

Limitations

- Limited by a sequential instruction stream
  - -Issue in extracting data and thread level parallelism
- -Its ability to process general applications costs time (as opposed to an ASIC unit)
- -Can compute everything, but not necessarily efficiently!
- = Performance limitations!
- Memory limitations!

Amdahl's Law

- Represents speedup of a program using parallel processors versus one serial processor

Amdahl's Law

- Represents speedup of a program using parallel processors versus one serial processor

$$speedup = \frac{wall\text{-}clock\ time\ of\ serial\ execution}{wall\text{-}clock\ time\ of\ parallel\ execution} = \frac{1}{S+\frac{P}{N}}$$

where S is the scalar fraction of the code,

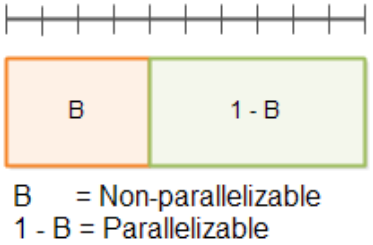
P is the parallel fraction

N is the number of CPUs or cores.

By normalization, $S + P = 1$

- A program may be split into 2 parts:
  1. Parts which can be parallelized
  2. Parts which can NOT be parallelized (serial)
- Therefore, S = 1-P

Amdahl's Law



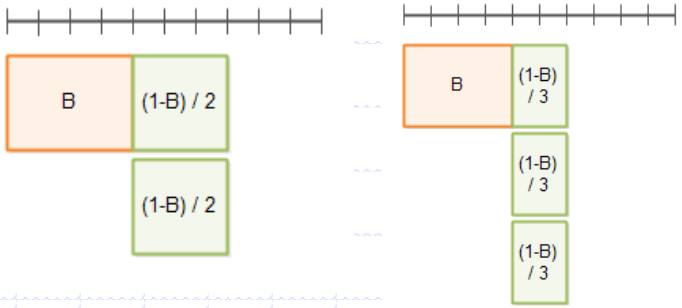
$$speedup = \frac{wall\text{-}clock\ time\ of\ serial\ execution}{wall\text{-}clock\ time\ of\ parallel\ execution} = \frac{1}{S+\frac{P}{N}}$$

where S is the scalar fraction of the code,

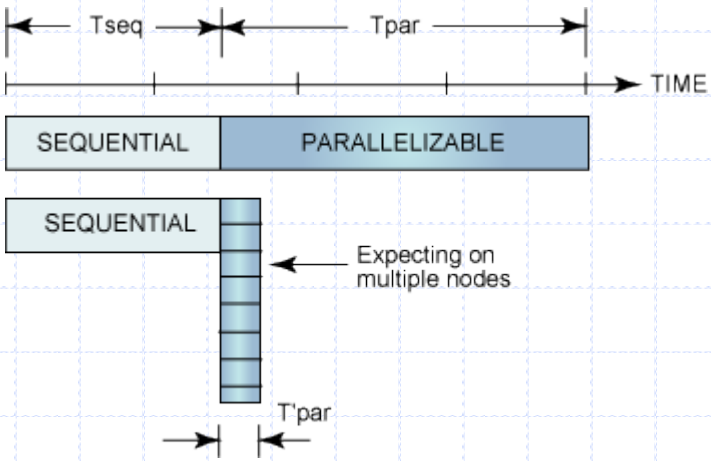
P is the parallel fraction

N is the number of CPUs or cores.

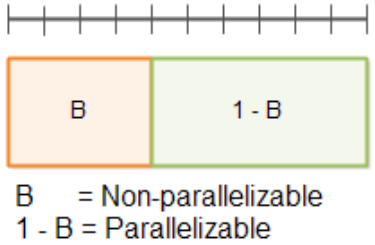
By normalization, $S + P = 1$

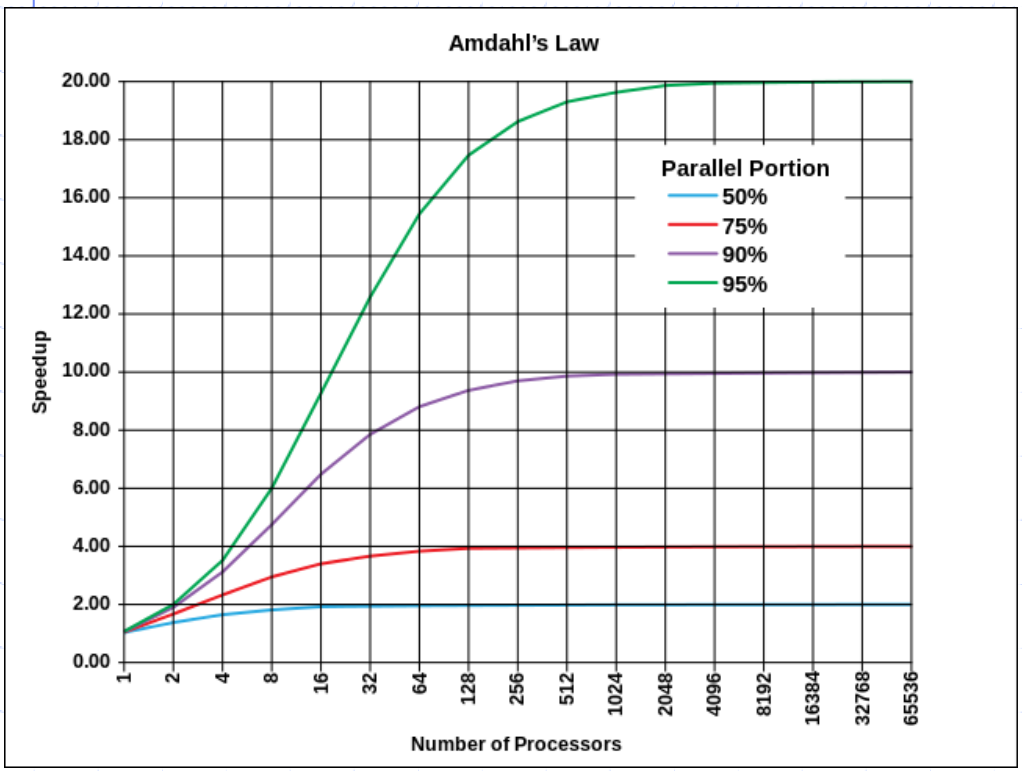


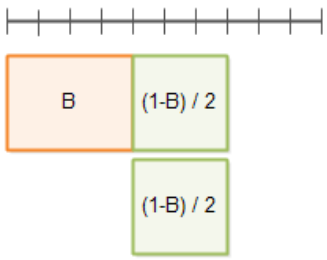
- A program may be split into 2 parts:
  1. Parts which can be parallelized
  2. Parts which can NOT be parallelized (serial)
- Therefore, S = 1-P

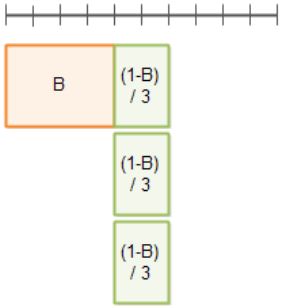


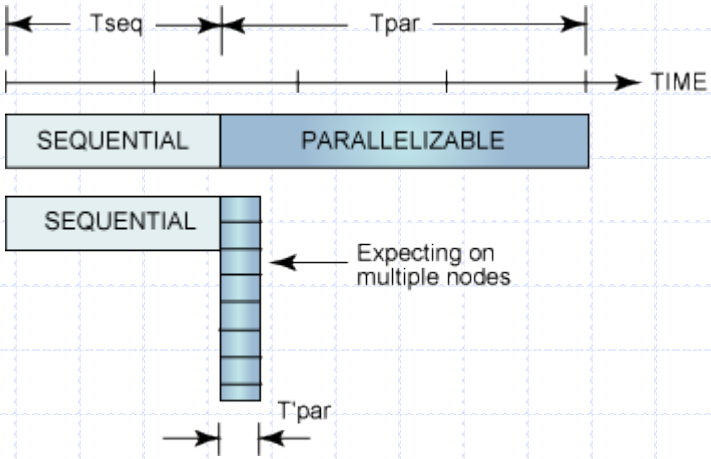
Amdahl's Law





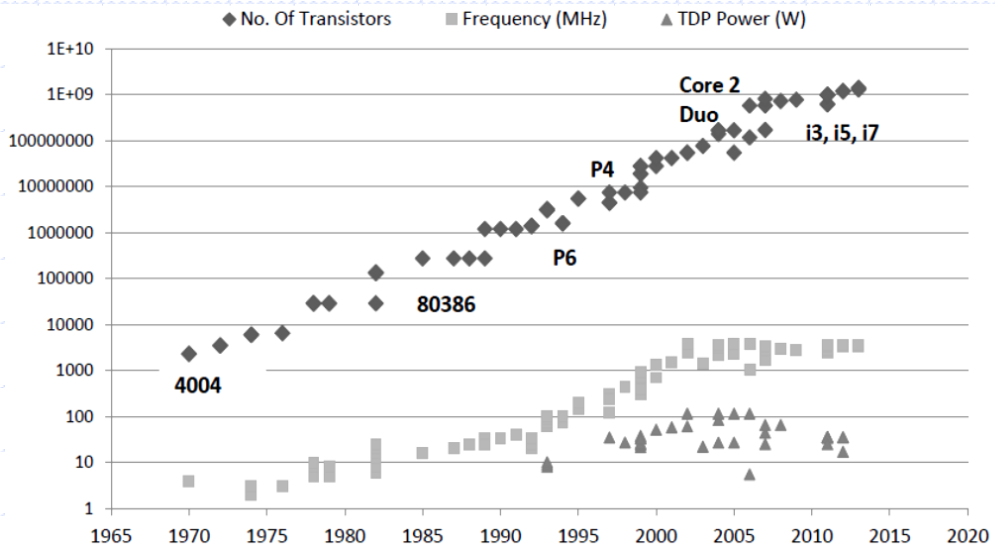






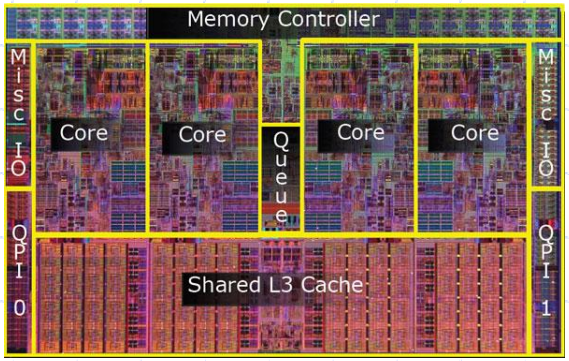
https://commons.wikimedia.org/wiki/File:AmdahlsLaw.svg
#/media/File:AmdahlsLaw.svg

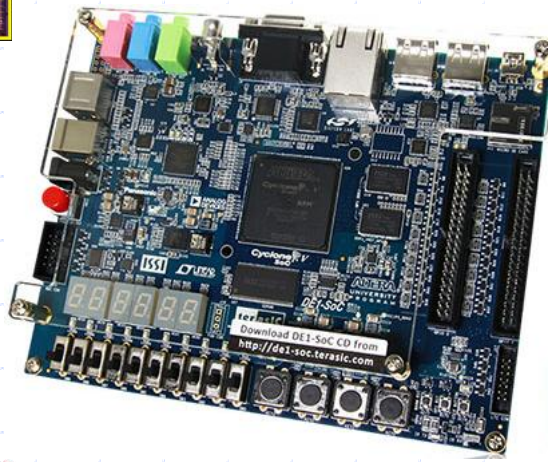
Moore's Law

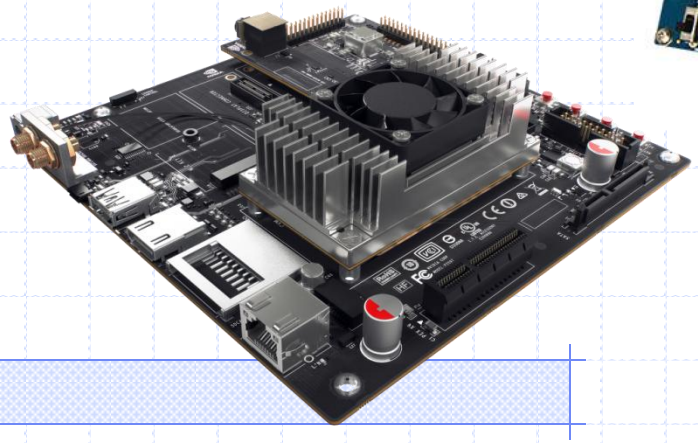


- Observation that the number of transistors in a dense IC (computing hardware) doubles approximately every 2 years

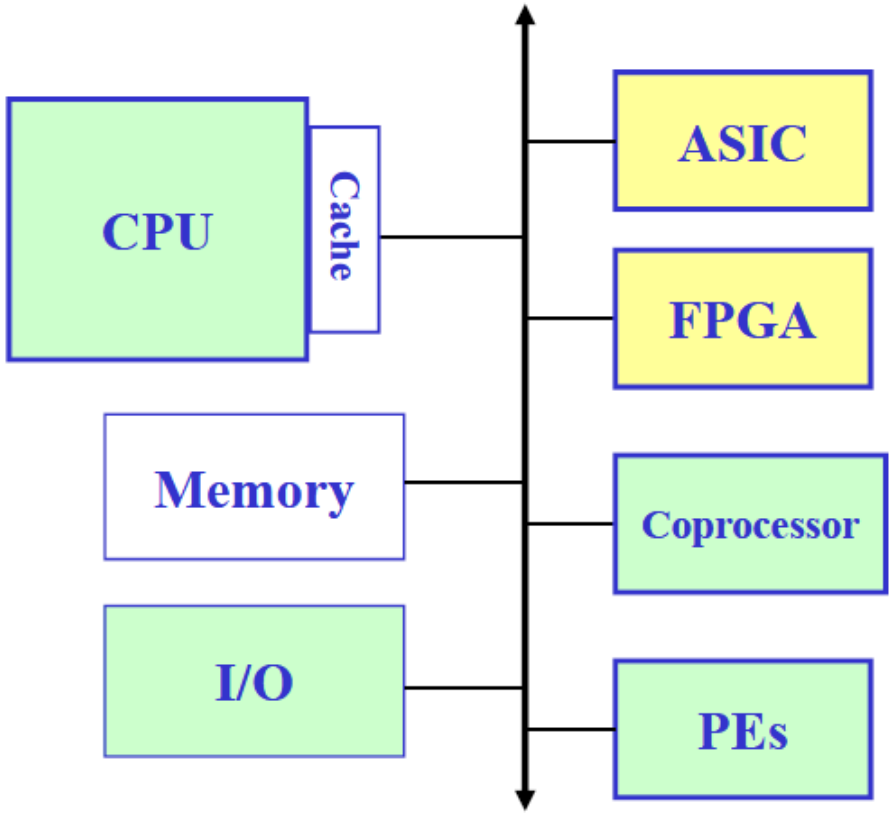
We're Heading Towards...





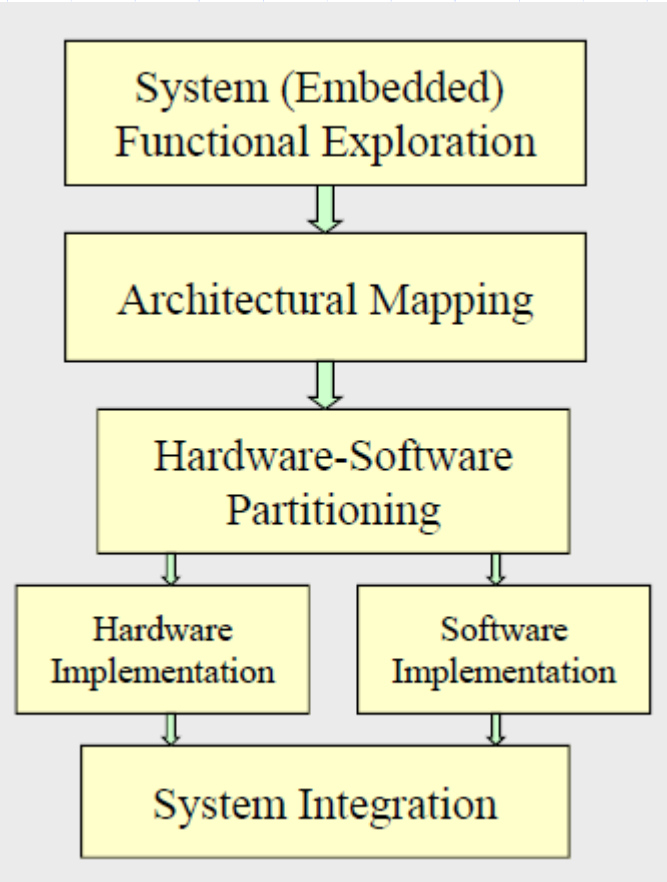


HW/SW in Embedded Systems



- CPU
- HW units
- Communication
  - Buses
  - NoC
- HW and SW?
- Interfacing?

HW/SW Co-Design/ Partitioning



- Define the product's requirements and produce a specification of the system
- Define the specifications as functions
- Partition the functions as silicon and code versions
- Map fts to code &/or silicon, determine costs – will your design meet the mandatory requirements of the system?
- Integrate, test, verify

HW Units – when's it worth it?

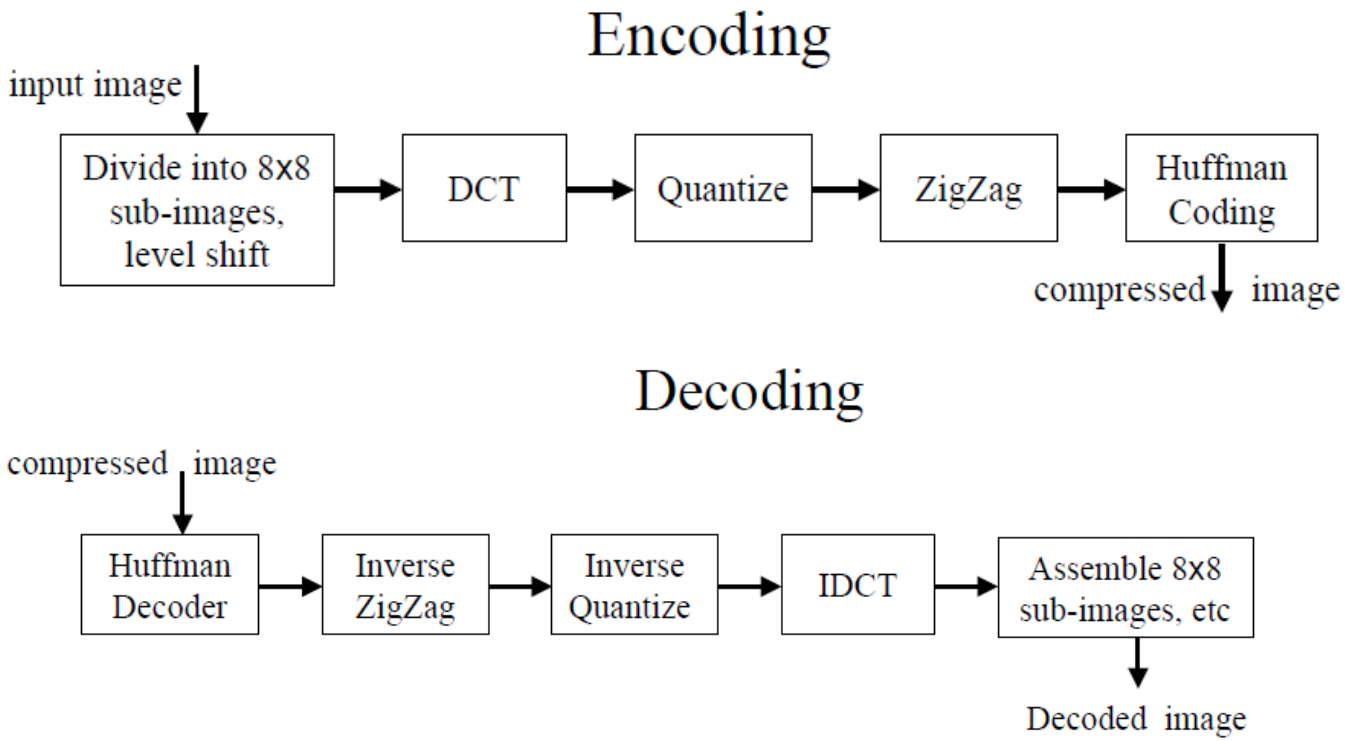
- Ideally, you should ask the following:
  - Which functions execute quickly on the CPU?
    - Then leaving it to execute as software is a good idea
  - Are there libraries that you keep using frequently?
    - Can possibly be integrated as co-processor hardware
  - Are there computationally intensive functions in the application (including libraries)?
    - Then they would likely benefit from offloading

HW Units – when's it worth it?

- Ideally, you should ask the following:
  - At what cost does the offloading come with?
    - What is the latency to write data, compute on the other hardware chip, and read back data
    - Do we need to revise the bus interconnect?
    - Will it actually be faster if the CPU just executes it taking into consider all the latencies?

Example

- JPEG Encoding and Decoding
  - Functional Specification:



Example

- Preprocessing --(Divide Into Functions)

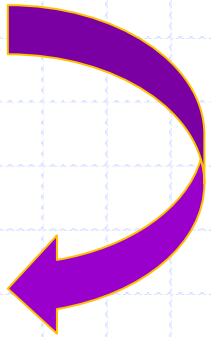
  dividing an image into 8 x 8 blocks pixel[i] = pixel[i] – 128 ;

- DCT

$$F(u,v) = \tfrac{1}{4}C(u)\,C(v)\left[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y) * \cos((2x+1)u\pi/16 * \cos((2y+1)v\pi/16\right]$$

where $C(u), C(v) = 1/\sqrt{2}$ for $u,v = 0$

$C(u), C(v) = 1$ otherwise



```
for (unsigned char uv = 0; uv < 64; uv++)
      input_data[uv/8][uv%8] -= (char) (pow(2,8-1));

for (u = 0; u < 8; u++) // do forward discrete cosine transform
  for (v = 0; v < 8; v++) { temp = 0.0;
            for (x = 0; x < 8; x++)
            for (y = 0; y < 8; y++)
              temp += input_data[x][y] * fcosine[x][u].read() *
                   fcosine[y][v].read();
     if ((u == 0) && (v == 0)) temp /= 8.0;
     else if (((u == 0) && (v != 0)) || ((u != 0) && (v == 0)))
       temp /= (4.0*sqrt(2.0)); else temp /= 4.0;
     out64[u][v].write(temp);
```

Example

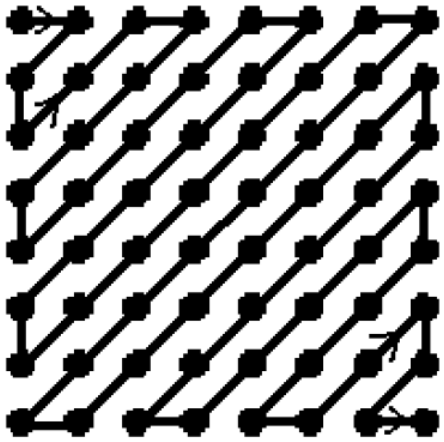
- Quantize

$$F_{quantized}(u,v) = F(u,v) / Quantization\_Table(x,y)$$

Quantization table :

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

- ZigZag



- 8x8 block (2d) → 1D 64 entry block
  - Orders values, places low-frequency coefficients before high

Example

- Huffman Coding -- compression

```
....

int bc, bn, ix;
for(i=0; i < BYTES; i++) {
            bc=0; bn=0;
            if ( efreqs[i] == 0 ) { codes[i] = NULL; continue; }
            ix = i;
            while( abs(preds[ix]) != ix ) {
                bc |= ((preds[ix] >= 0) ? 1 : 0 ) << bn;
                ix = abs(preds[ix]);
                bn++;
            }
            codes[i] = malloc(sizeof(huffcode_t));
            codes[i]->nbits = bn;
            codes[i]->code = bc;
}

Some other IF conditions
....
```

Example

- Good function candidates for Hardware??

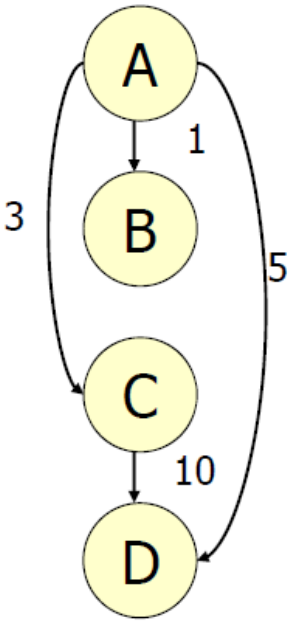
- What is ideal for software execution?

Example

- Good function candidates for Hardware??
  - DCT contained many loops, all independent. Most computationally intensive function
- What is ideal for software execution?
  - Rest of the functions were pretty lightweight

Mapping

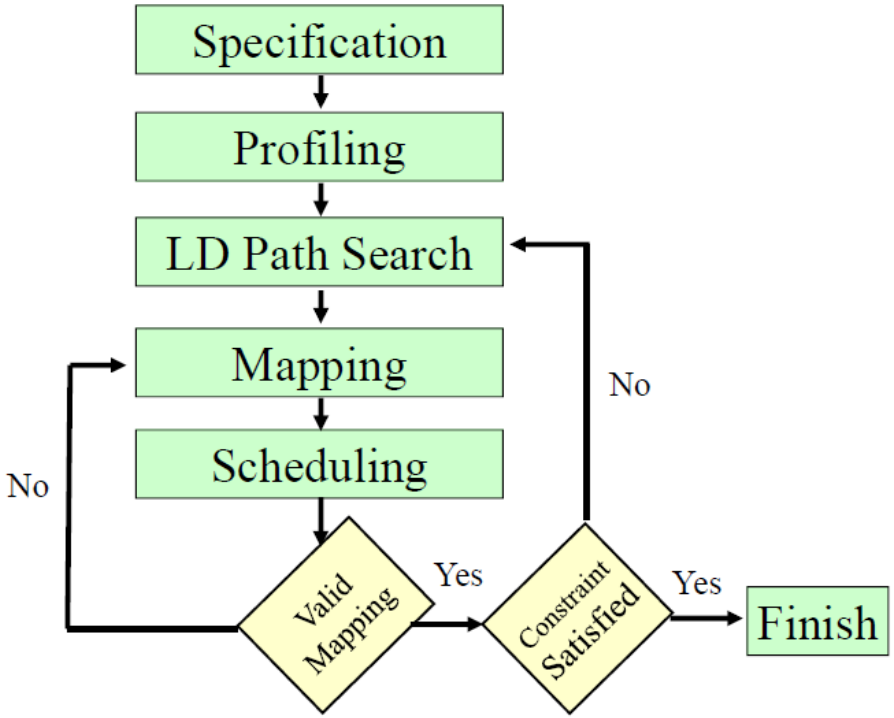
- DAG: Directed Acyclic Graph
  - Models basic functions/blocks of the system
  - Describes the order (dependence and precedence) of the functions in the system
    - Also exposes any parallelism in the system
  - Graph is composed of vertices and edges
    - Vertices = functions/task
    - Edges = dependency
    - IF weighted edges = communication time

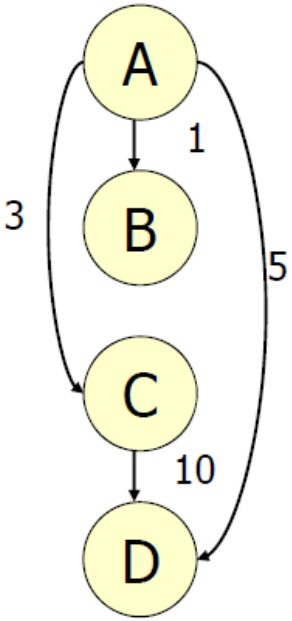


Mapping

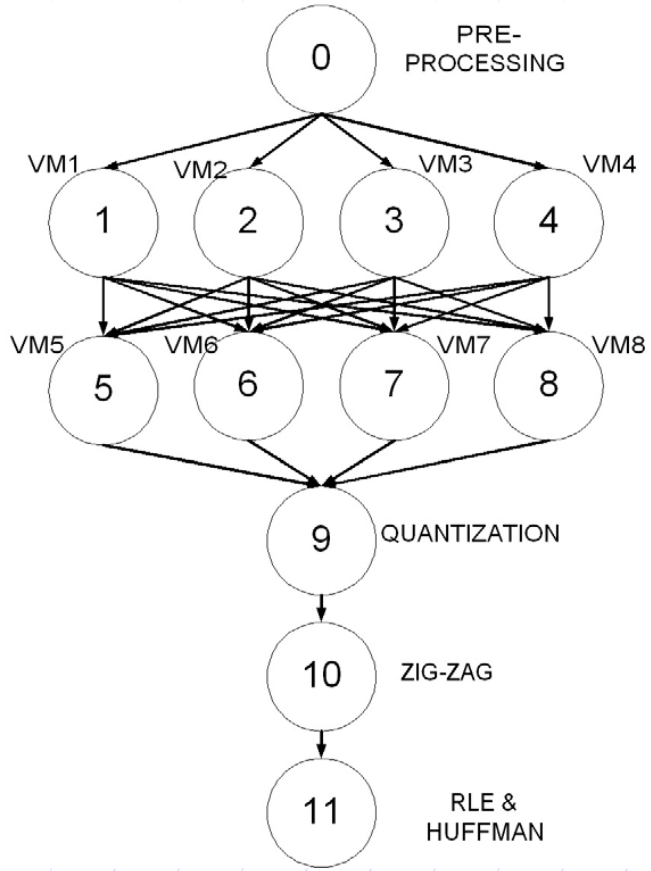
- Map each vertex as HW or SW
- See if mapping is valid and constraints are satisfied

DAG-based Partitioning Structure



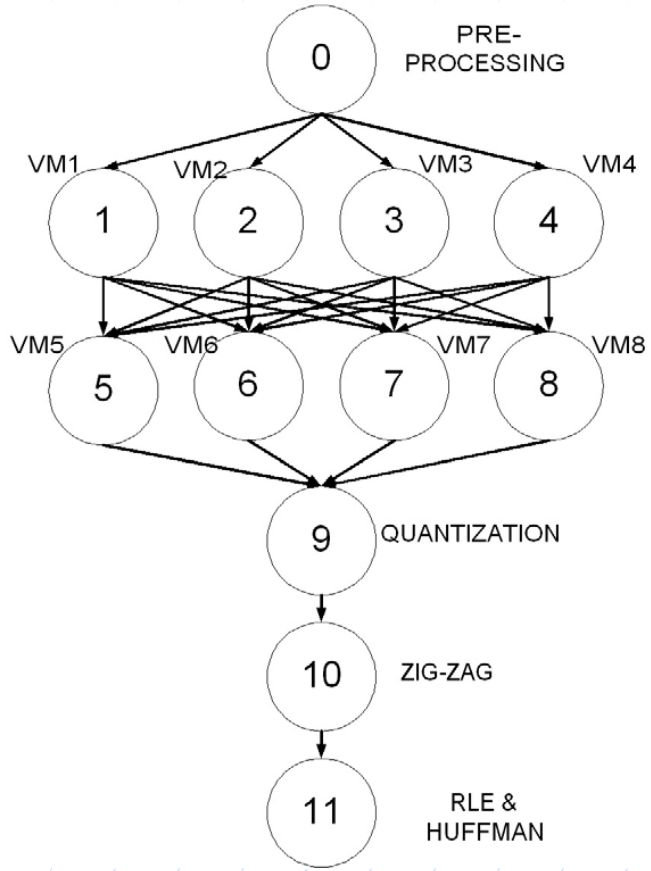


Example -- System Requirements

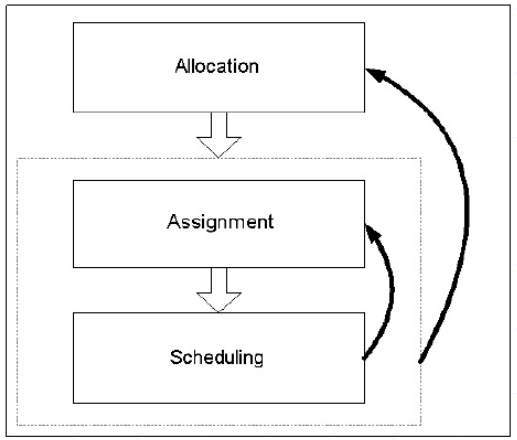


- Should:
- Encode a JPEG in less than 12000 usec
- Consume 50000 transistors or less for the dedicated HW units
- Power → ...

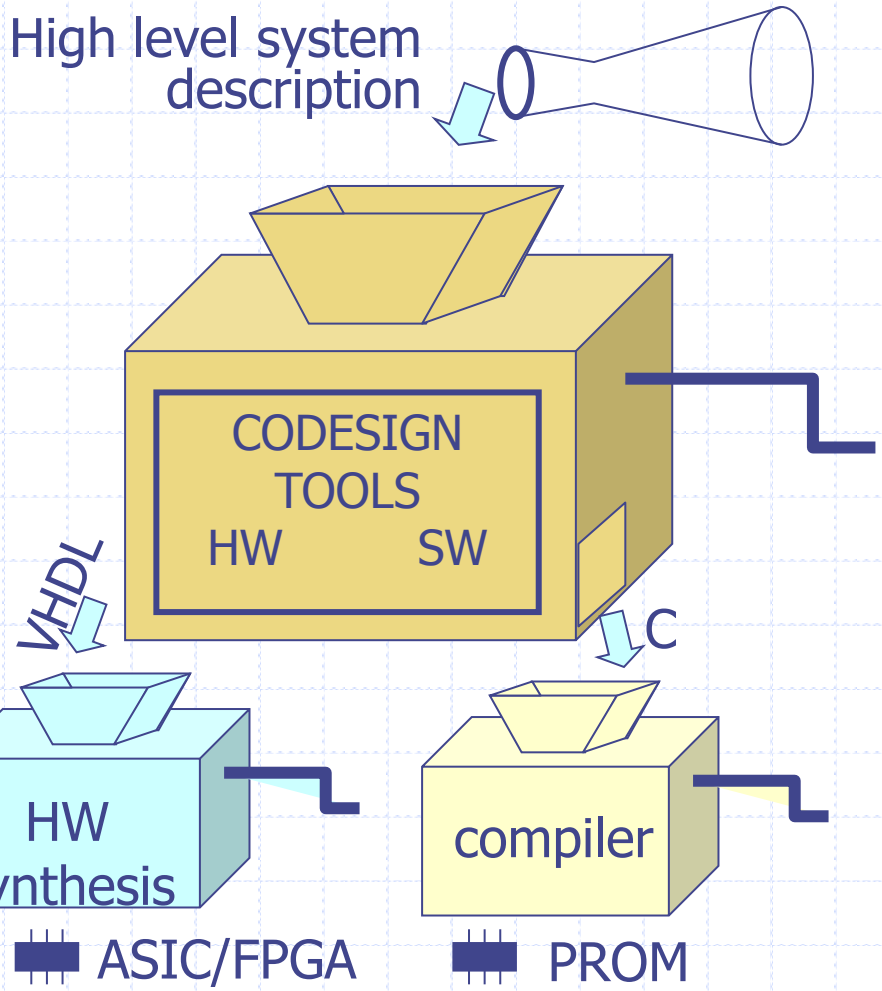
Example -- System Requirements



- Creating a software tool, or using CAD tool to map out possibilities
  - HW/SW Co-Design Tools
- If you want the utmost optimal = exhaustive search



HW/SW Co-Design Tools



HW/SW PARTITIONING EX...

HW/SW Interfacing

- How does the hardware access the software? And vice versa?
- In general CPU-based embedded systems:

HW/SW Interfacing

- How does the hardware access the software? And vice versa?
- In *general CPU-based* embedded systems:
  - **IRQs** – communicates ready data (hardware, IOs etc) to the software
  - **Variables/registers** – software holds values in hardware memory

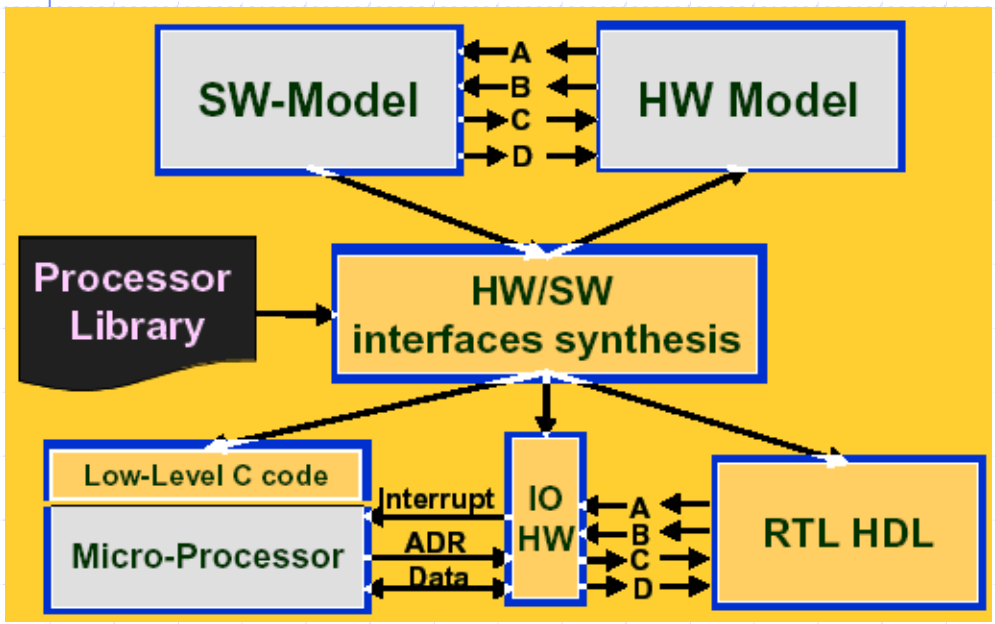
HW/SW Interfacing

- How does the hardware access the software? And vice versa?
- In accelerator-based embedded systems:

HW/SW Interfacing

- How does the hardware access the software? And vice versa?
- In accelerator-based embedded systems:
  - IO Addressing (ports) and
  - Memory Mapping

HW/SW Interfacing

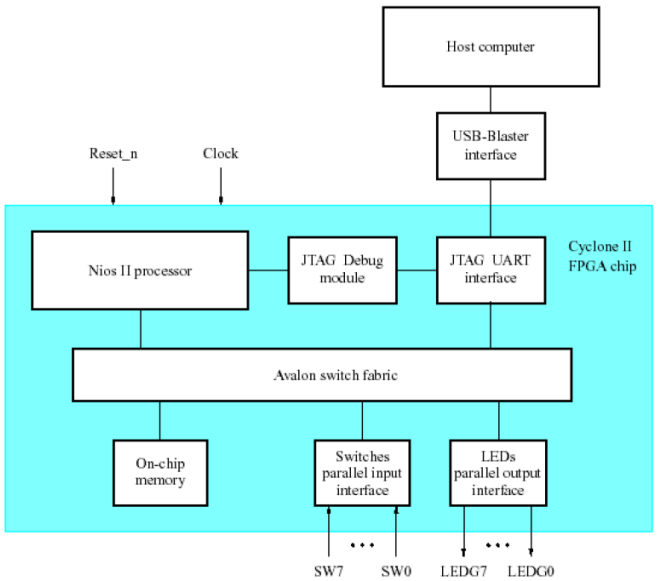


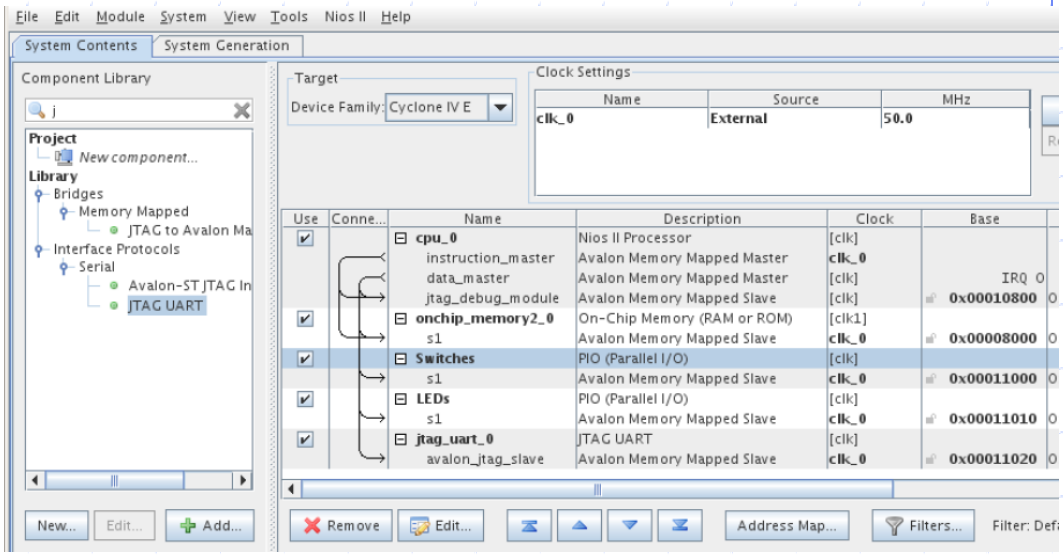
- IO Ports
  - If any HW unit is included in the system, it must be connected to the main uP
  - How?
  - Theoretically: IO ports
  - Practically?

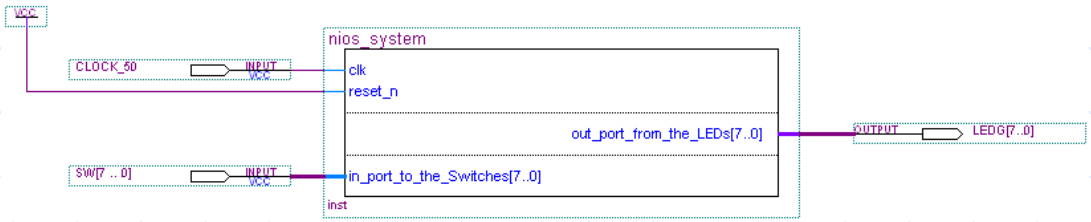
HW/SW Interfacing

- Once connected, the software must be provided with a device address
  - To send and receive data, control signals etc
  - = **Memory mapping**

Bonus Lab – HW/SW I/F








```
#include <stdio.h>
int main()
{
  unsigned char * Switches;
  unsigned char * LEDs;

  Switches = (unsigned char *)0x0011000;
  LEDs = (unsigned char *)0x00011010;

  printf("Beginning.\n");

  while(1){
    *LEDs = *Switches;
  }

  return 0;
}
```

Side Note: for General CPUs

- How is offloading (or workload sharing) done for general CPUs?

OpenMP
MPI

OpenCL

```
__kernel void
  matrixMul(__global float* C, __global float*

    int tx = get_global_id(0);
    int ty = get_global_id(1);

    float value = 0;
    for (int k = 0; k < wA; ++k)
    {
      float elementA = A[ty * wA + k];
      float elementB = B[k * wB + tx];
      value += elementA * elementB;
    }
    C[ty * wA + tx] = value;
}
```

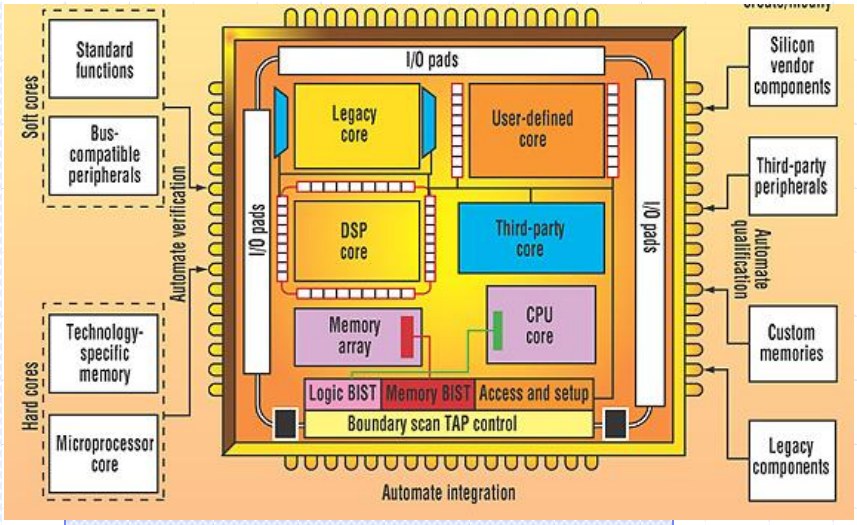
```
__global__
void saxpy(int n, float a,
           float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

int N = 1<<20;
cudaMemcpy(x, d_x, N, cudaMemcpyHostToDevice);
cudaMemcpy(y, d_y, N, cudaMemcpyHostToDevice);

// Perform SAXPY on 1M elements
saxpy<<<4096,256>>>(N, 2.0, x, y);

cudaMemcpy(d_y, y, N, cudaMemcpyDeviceToHost);
```

CUDA

OmpSs

System-On-Chip (SoC)

- An IC that integrates all components of a system onto a single chip
  - Versus CPU and an external device
  - i.e. SoC contain CPU(s), GPU, memory, USB, power management ICs, 3G modules etc all on a single die



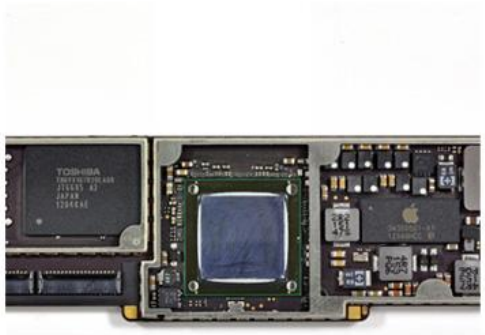
System-On-Chip (SoC)

- An IC that integrates all components of a system onto a single chip
  - Versus CPU and an external device
  - i.e. SoC contain CPU(s), GPU, memory, USB, power management ICs, 3G modules etc all on a single die
  - Several CPUs are now actually considered SoCs!?

System-On-Chip (SoC)

- Several CPUs are now actually considered SoCs!?
  - CPUs now contain the CPU itself, along with integrated graphics processors, PCI express, memory controllers etc all on a single die

PC Motherboard – CPU with support ICs



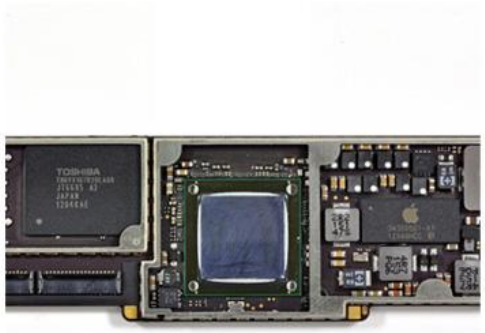
Ipad3's CPU SoC circuit → A5

System-On-Chip (SoC)

- Several CPUs are now actually considered SoCs!?
  - CPUs now contain the CPU itself, along with integrated graphics processors, PCI express, memory controllers etc all on a single die

PC Motherboard – CPU with support ICs

Advantages?
Disadvantages?

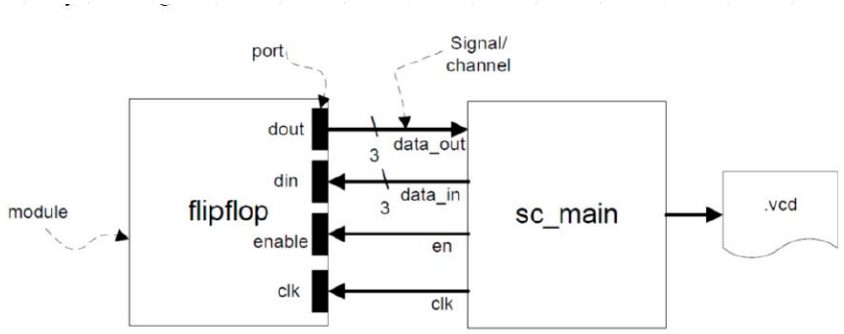


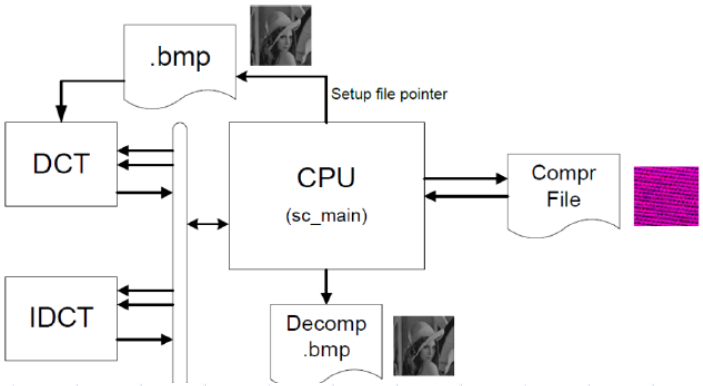
Ipad3's CPU SoC circuit → A5

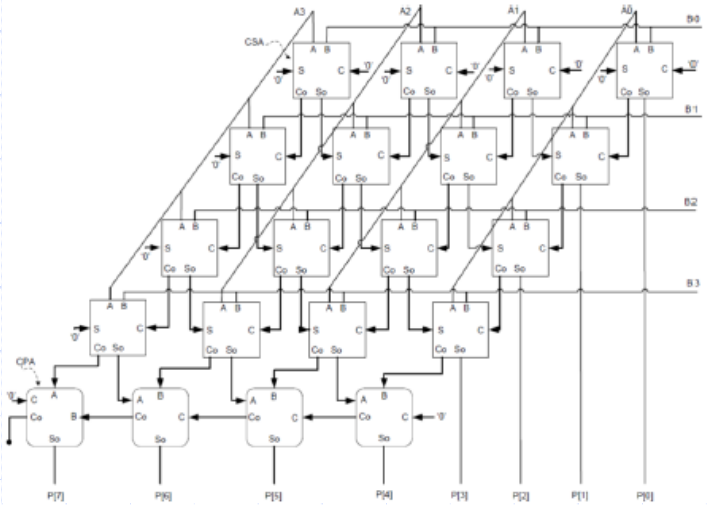
System-On-Chip (SoC)

- An IC that integrates all components of a system onto a single chip
  - Versus CPU and an external device
  - i.e. SoC contain CPU(s), GPU, memory, USB, power management ICs, 3G modules etc all on a single die
  - Several CPUs are now actually considered SoCs!
- Basis for all embedded systems
- Uses IP cores, integrates them on a chip to increase productivity

COE838: System-On-Chip

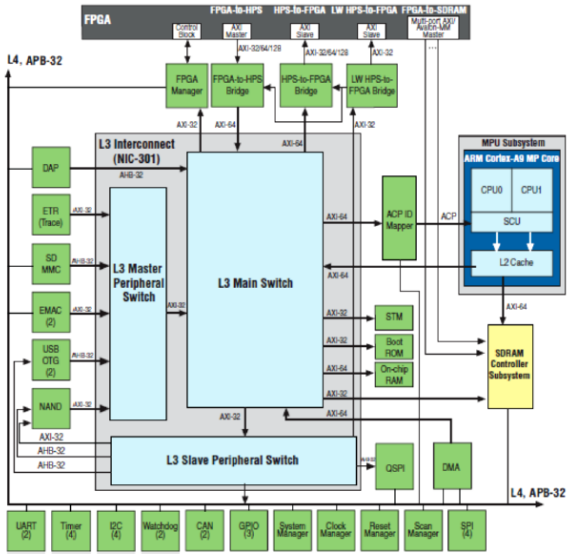


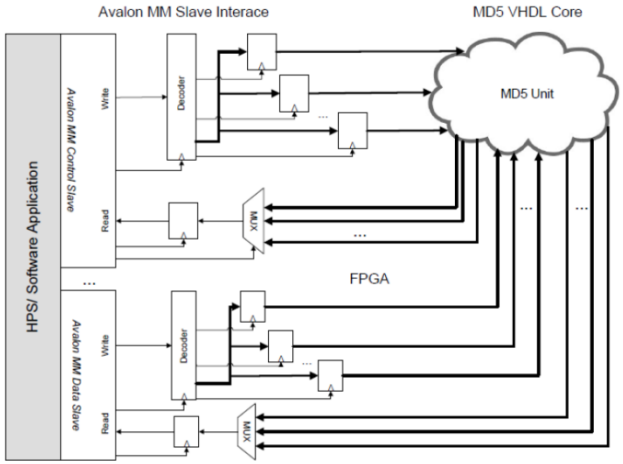


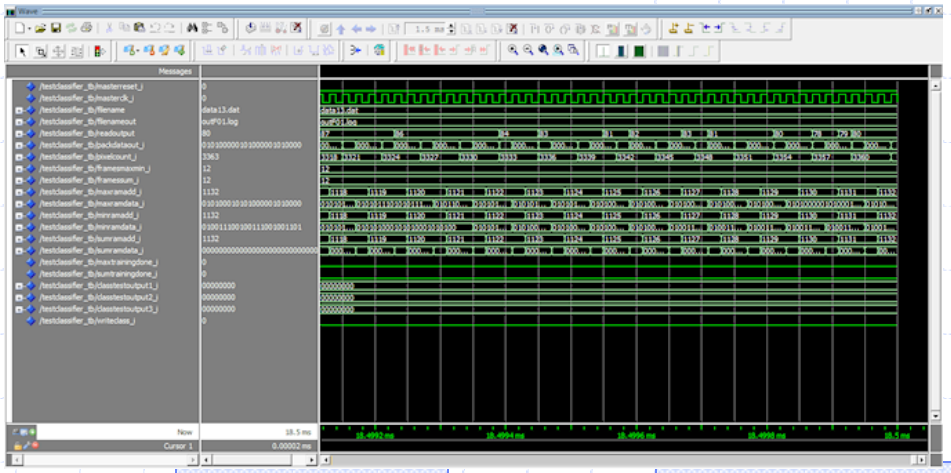


- **SystemC** – model HW/SW systems using C++ extension
- Simulation of accelerators and CPUs

COE838: System-On-Chip







- **HPS/FPGA Systems** – ARM Cortex-A9 and Cyclone V FPGA
- Interfacing, emulating, simulating real HW/SW systems

COE838: System-On-Chip

Next week….

- General review
- Take up practice exam
  - Will post on the weekend/ Monday latest