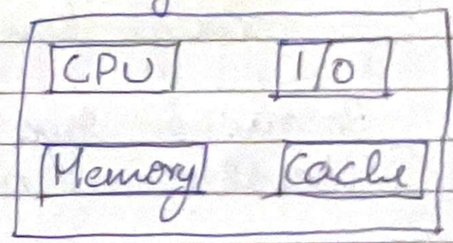
Embedded System Modeling.

→ Lecture1

Computers are everywhere

System - on - chip (SoC)
- Design of a complex embedded systems on a single chip
- reduces cost
- easier to build the system



Systems embedded into cyber-physical systems
- Contrains for external ip (real time)
- Application specific (not general purpose)

challenges to design.

1) often mobile (Battery powered)
2) highly reliable (can stand ↑ temp)
3) high performance constaints (are often real-time)

4) High complexity.

Advantages

- application known at design time since it is app specific.
- environment known at design time.
- Customization.

SoC design faces tremendously high design complexity.

Abstraction layer.

Level	no of components	
System	$1e^{0}$	Abstraction ↑ / Accuracy ↓
Algorithm	$1e^{1-2}$	
RTL	$1e^{3-4}$	
Gate	$1e^{5-6}$	
Transistor	$1e^{6-7}$	

Move to higher level of abstraction.

Product features
↓
Specification model
↓
Architecture model
↓
Communication model
↓
implementation model

} top-down system design flow.

Model

It is an abstraction of reality
- some features explicitly present
- simplified feature
- some features are omitted

Modeling

It is the creation of model.
The refinement of model.

Model of computation.

It has an underlying mathematical model behind it.

1) Programming models.

- C/C++, Java, python

2) Process based

- Processes and threads
- Kahn Process model
- Synchronus data flow

3) State based models.

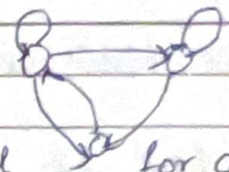
- FSM, DFG, FSMD, SFSMD, HCFSM

Computational model

* Formal, abstract discription of a computing system

with multiple
- supported features
- complexity
- expressing power

- FSM



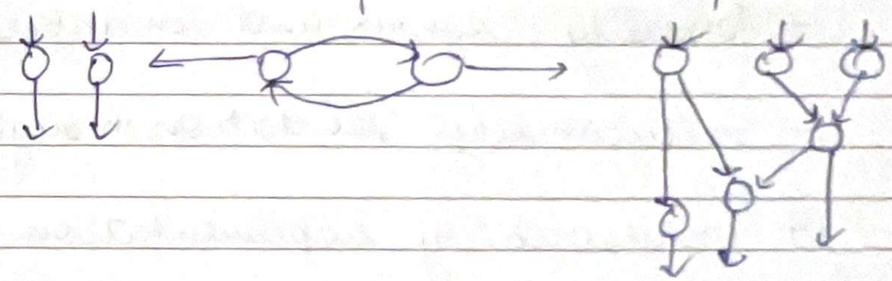
Basic model for describing control

- DFG Data flow Graph

Basic model for describing computation of data flow.

- FSMD finite state machine with data

Combined model for control & computation



- SFSMD Superstate FSM with data

States are described by procedures in programming language

- HCFSM Hierarchical concurrent FSM

multiple FSM composed hierarchially and in parallel

- Program state Machine (PSM)

States described by procedure in a programming language.

eg: Spec C

System level Description Language.

- Goals and requirements.

- formality: Syntax and semantics.
- Executability: Validation through simulation
- Synthesizability: implementation of HW & SW.
- Modularity: Seperation of concepts
- completeness: support for all concepts found in embedded system.
- Orthogonality:
- Simplicity

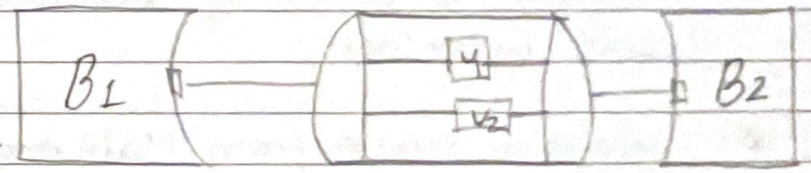
SLDL - orthogonal Concepts & constructs

(SLM) System level Modeling
- Computation encapsulated in behaviors
- Communication encapsulated in channels

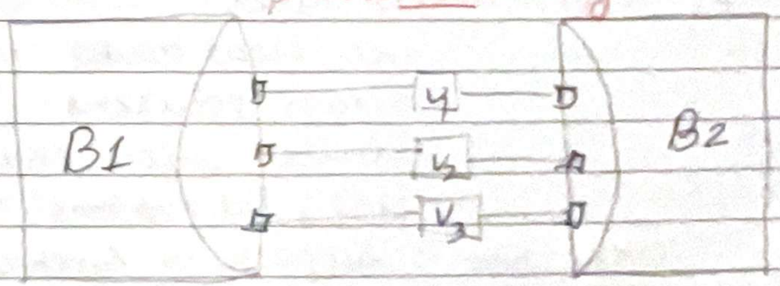
Spec C

* Behavior and channels
* Seperation of computation & comm
* plug and play

- Specification -exploration model



- implementation model



Channel disappears
Communication inlined into behavior
Wires are exposed

→ Lecture 3.

Spec C language

* foundation : ANSI-C

- Spec C is a true superset of ANSI-C
- every C program is a spec C program

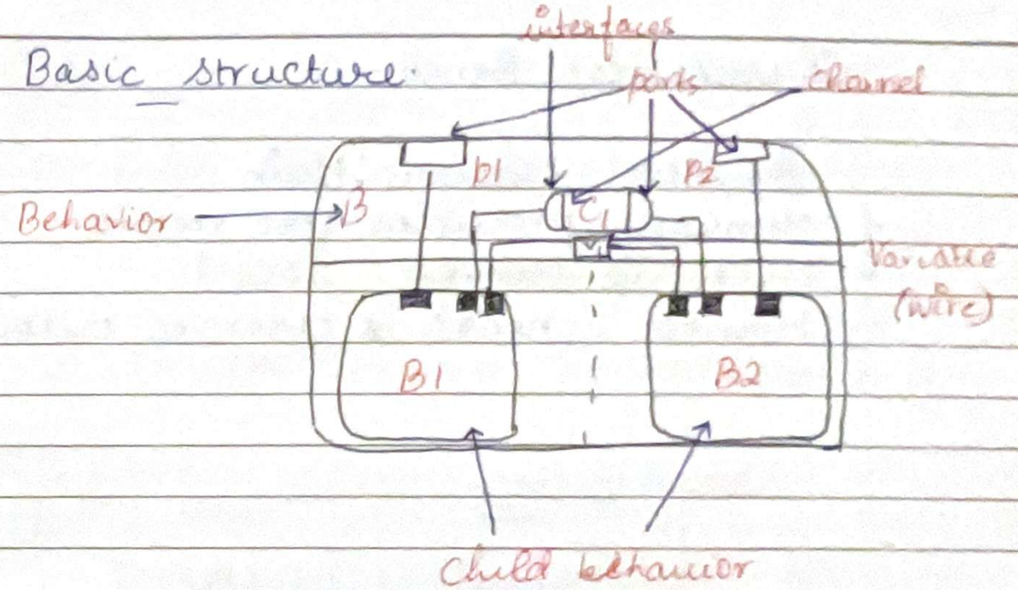
- Spec C has extensions needed for hardware
- Spec C is a real language.
  relies on dedicated compiler & static Analysis

* Program is a set of behaviors, channels and interface

* execution starts from Main.main() behavior.

* Supports all types.
  int, float, double.
  arrays, pointers
  struct, union, enum
  bool, bit vectors
  event type : support of synchronisation.
  event e.
  Buffered & signal type : explicit support of RTL concepts.

Basic structure:



structural hierarchy

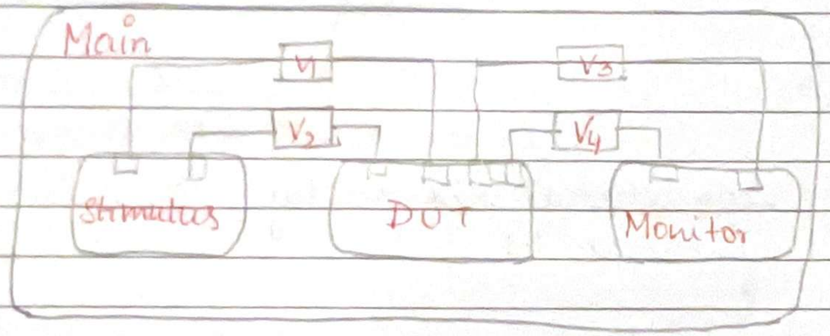
```
interface I1
{
  ;
};

channel C1 implements I1
{
  ;
Behavior B1 (          );
behavior B (           );
{

    void main ()
    { par { b1;
            b2;
          }
    }
};
```

Typical test Bench

- Top level behavior : Main
- stimulus provides test vectors
- DUT is target SoC
- Maintor checks & observes output



Behavioral hierarchy

* Sequential execution
* FSM execution
* Concurrent execution
* pipelined execution

behavior B_seq
{B b1, b2 b3;
void main()
{ b1
b2
b3
};

behavior B_fsm
{ B
void main()
{ fsm { b1
b2
b3
} }
};

B_par
{
{ par { b1
b2
b3
} }
};

B_pipe
{
{ pipe { b1
b2 }
}
};

Pipeline

- Explicit execution in pipeline fashion

pipe { <instance_list> };

pipe (<init>; <cond>, <incr>) { }

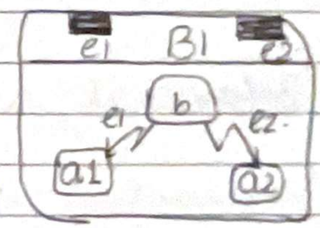
- Supports automatic buffering

piped [...] <type> <variable_list>;

for ref code check slide '21' pg # of pdf

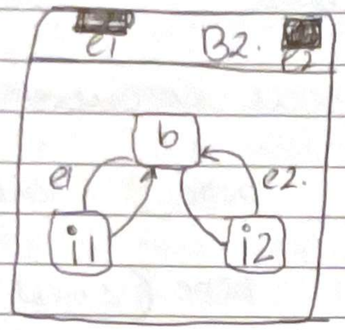
Exception handling

- Abortion



behavior B1(in event e1,
in event e2)
{ B b, a1, a2
void main (void)
{ try { b; }
trap (e1) { a1; }
trap (e2) { a2; }
}
};

- interrupt

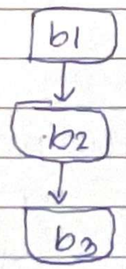



```
Void main (Void)
{   try { b; }
    interrupt (e1) { i1; }
    interrupt (e2) { i2; }.
}.
```

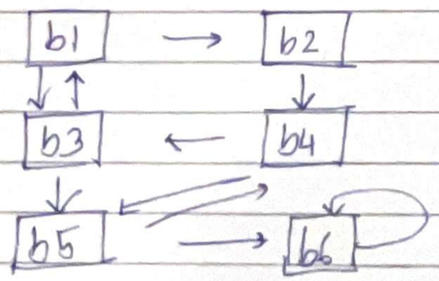
→ Lecture 4 Spec ~~System~~ C

- Behavioral hierarchy.

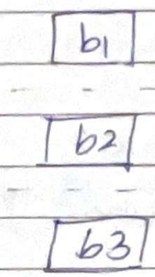
1) Sequential execution



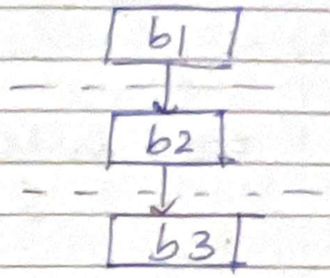
2) FSM.



3) Concurrent



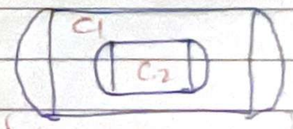
4) pipeline



- Communication and synchronisation

Message passing { - Via shared Variable } Shared mem
- Via channel & interfaces.

can also happen
- via hierarchical channel.



protocol stack

Synchronisation happens ~~to~~ using

- wait
- notify
- ~~Ack~~

Behavior/Event wait for a request and then wait for 'ack' once it is sent by the other event

* how to define a channel or interface.

⇒ interface <name>
{ < declaration > }.

⇒ channel class

channel < name >

implements < interfaces >
{ < implementation > }.

* When using shared memory, you need Semaphores.

→ Semaphore
~~Synchor~~
Synchronisation ~~parameters~~. Primitives.

- It is a unsigned interger.
- changes are atomic.
- can only have 2 operations.
  Wait()
  post().

Wait()

It decrements the semaphore, unless it's value is 0

When it 0, it waits.

Post()

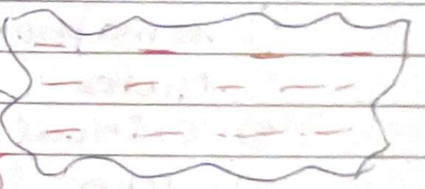
Increments the value.

Binary Semaphore.

↳ They act as locks.

↳ Value 0 or 1.

Wait ( ) { → decrements.



} post() → increments

Critical Section

→ Mutually exclusive executions.

1) Semaphores.
2) Mutex.

* Single handshake is for synchronisation.
* double handshake is for communication.

→ standard channels.

* Import synchronization channel.

1) Semaphore
2) Mutex.
3) Critical section
4) token
5) barrier.
6) Handshake

* type - less communication channel.

1) double handshake.
2) Queue.

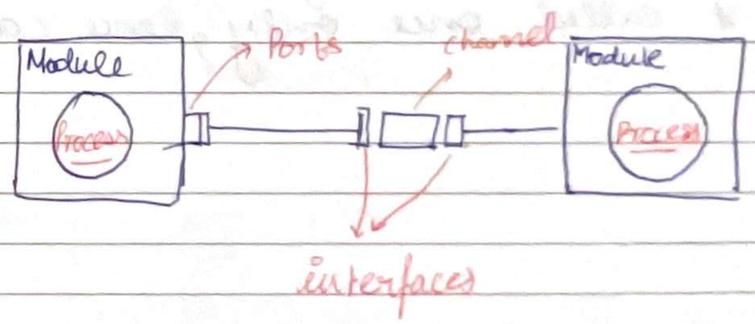
* typed communication channel

1) double handshake.
2) queue

→ Lecture 6 — System C 2.3.1 ↳ recent

* developed from C++
* Abstraction levels

- untimed model.
- Transaction level model.
- Bus functional Model.
- Cycle accurate model.



ports are bidirectional, since no direction is mentioned

Systemc R → library.

SC_Module { → class.

→ Process in Systemc

SC-method.

* fast or 0 - time.
* can not contain infinite loop.
* Can not be suspended.

SC-thread.

* Slower, but powerful.
* can have infinite loop.
* called once only, hence can have ∞ loop.

→ Lecture 9

Essential concepts in embedded system Modeling

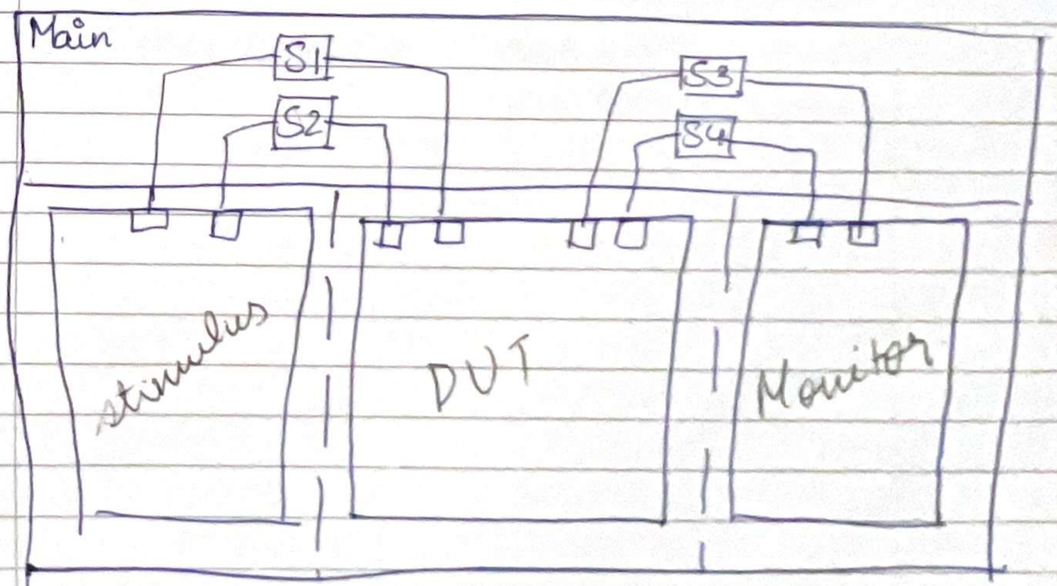
- Behavioral hierarchy
- Structural hierarchy & connectivity.
- Synchronisation and Communication
- Timing.

→ Lecture 10

Test Bench Modeling.

Create

→ Stimulus
→ DUT
→ Monitor.



* Bus functional Model

* We simulate the test bench, but dont synthesis it.
* We simulate & Synthesis the DUT.

~~Lecture 12~~

→ Lecture 12

Computation

Granularity – leaf behavior
hierarchy – Explicit execution order
encapsulation – localised variable
explicit port mapping
concurrency – Potential parallism
timed – untimed

Communication

Comm – std channel lib.
Synch – 〃
Dependencies – data flow explicit
in connectivity

How to improve speeds

→ pipelining

→ Parallelization

→ hardware optimisation

→ Software "

→ app "

→ Lec 17

* Real time goal — No. } A7
how far off — 211x. }

1/3890 = 257 FPS

Assignments Assignment

→ A1

Goal: Design a suitable embedded system model of this application & describe it using SLDL

1) Bug fix in NMS.

1 col and row is being omitted, alter that

2) All Warning must be cleared as we go. so that they do not form problems in the future work.

Warning can become a bug later.

→ A2

1) Create prod cons behavior which are parallel

→ A3

1) prod - cons in SystemC.

SoC supports initialisation of variables with expression that are constants at compile time.

→ A4 ↓ creating the SL specification model

1) The Macro NULL is not defined, so change it to zero '0'

2) Flexible configuration parameter should be converted to constants

Compile time constants: malloc ()
calloc ()
free ()

simpler ↑ code, reduce ↑ memory fragmentation → not feasible during hardware implementation.

↓ Memory, ↓ increase performance

☆ SoC can not be represented as a new memory chip at run time.

☆ Command line para. can only be used with test bench, not the SoC. eg: filename

3) Window size = 21

sigma, rows, cols.

max sigma = 4 ← replace with 4

WindowSize = 1 + 2 * Ceil (2.5 * Sigma)

= 1 + 2 * Ceil (10)

= 1 + 2 * 10

= 21

improved performance, safer code, easier debugging

4) radian direction ≠ angle radians

↳ radiant direction image

Since we only need edge image, we do not require these 2 functions.

→ A5

1) convert application to process video instead of picture

2) pixel to (2704 × 1520) produced by the drone.

↓

higher memory for our application

↓

array variables holding our data

3) adjust stack overflow

Otherwise get segmentation fault

We adjusted stack space allocation in linux.

stack size → 128 ~~KB~~ Mbytes.

4) Create test bench.

☆ Stimulus, platform, Monitor (in parallel)

data in — DUT — data out.

Better integration & testing platform can abstract away the complexities of handling diff dt input types

* data in & data out, allow our test bench to remain un-modified

even when communication to DUT is implemented by a BUS

* Our target SoC will never stop, we run an endless loop. unless power is turned off.

→ A6 start designing our target implementation.

1) all function become behaviors.

* Sequential execution. } Reduce pipeline idea starts. } battle neck

2) refine gaussian smooth } Battle neck

→ A7 Performance estimation

* Calculate the time using <time.h>

got time in seconds & then convert to %

SystemC

↳ because sysc is pipelined by default

→ A8

1) Track the delay between frames in ms

2) Add delay for R-pi, to replicate Real time values.

Real time embedded values

3) calculate the throughput (FPS)

We can operate on multiple images

4) pipeline DUT

5) Decompose GS

↳ Blur X

↳ Blur Y

Battle neck start exits

After decomposing Blur Y is the Bottleneck

6) Slice Blur x and Blur Y

parallelise the functions

Now, NMS becomes the BN

→ A9

1) Optimisation —O.

* We dont always write optimised code because sometimes compiler may remove important part of the code.

2) ~~2x~~ 25x speedup after optimisation.

3) R-pi 3 to R-pi 4

↳ Better hardware.

4) floating point to fixed point
→ faster execution speed → cheaper ↳ faster
NMS → Bottleneck. ↳ with acceptable loss of accuracy
→ floating point was too slow.

* To ↑ throughput & by ↓ load on pipeline stages.

* need to optimise long stage delay.

65KB

We shift, so that int & fixed can represent fixed values with appropriate range

Make file may fail as it expects an exact match for generated img against reference image

* 320 - 290ms
↳ fixed pt

* SystemC is diff from SpecC & C++

ppt → Lec 3

SystemC & C++

1) hierarchical modeling
2) Process & threads.
3) modules & channels.
seperation of computation & communication.
4) explicit modeling of time
5) primitive channels.
6) concurrency
7) Synchronisation

R-pi3 or R-pi4 } • Real-world performance,
• physical prototyping platform
• Reflect Real-world

__/__/__

Final (A9)

→ What can we do to increase speed.

→ kip improving bottleneck

→ lower image resolution

→ accept lower frame rate.

→ Hardware changes etc

→ Optimization. (A9)

1) debugging.

2) Can increase size of compiled code, which can be a problem in ENV with limited memory.

3) Can increase computation time sometimes.

4) not compatible for all hardware.

__/__/__

A4) So many parameters are converted into fixed/constants. Why?

1) increased performance
2) lower the memory usage
3) reduce memory fragmentation } dynamic
4) easier debugging
5) less computation, less error possibility.

* Real time embedded system tries to use less resources.

RTES
→ less flexibility of memory, processing power & i/o operations.

malloc → static
because we can not create new memory at run time.

* We change this in ~~HW~~ SW so that it matches HW situations.

A5) After increasing the number of pixels the stack size would become larger.

↳ Variables / array ↑
* which get allocated on a stack.

→ pipeline

A8) instead of waiting for the blur op to complete on 1 image before starting the next op, we can pipeline and start the next op while the current one is still being processed

changes in delay

When profiling an application with the GNU profiler on a shared server, the results can vary due to users running heavy application.

because CPU, memory usage are shared among multiple users & applications running concurrently on the server.

heavy applications running will consume large amount of resources leading to other application being profiled.