Assignment 1

Professor Gunnar Schirner

Aly Sultan

ECE Department

HWSW Codesign

Table of contents

Problem 1.1

During design space exploration as part of the system design process, the target system architecture and its key architectural parameters are decided on. These design decisions have a major influence on the final design quality metrics such as (i) performance, (ii) power, (iii) cost, and (iv) time-to-market: (a) Briefly discuss how the following target platform styles rate in relation to each other in terms of the metrics listed above:

a)
- A pure software solution on a general-purpose processor
  - Fastest time to market due to relatively shorter development cycles and lower verification overhead
  - Power consumption will be set by general purpose processor features. It may be likely that in order for a viable pure software implementation the design must include a relatively overpowered processor which increases power consumption dramatically.
  - Cost is generally cheaper given the widespread availability of general purpose processor ips and processors chips in the market (ignoring any increases in cost associated with using a more powerful processor) Development does not incur as much overhead as other methods given that there is a lower variety of developers required in smaller numbers.
  - Lowest performance if the nature of the problem being solved is parallelizable. This can be overcome slightly with more powerful general purpose processors that include SIMD instructions, however as discussed above, those types of processor result in increased power consumption and higher costs.

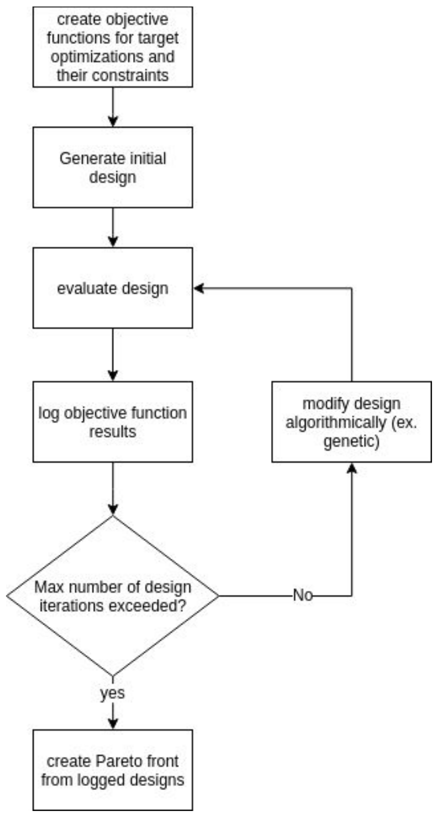
- A general-purpose processor assisted by a custom hardware accelerator/coprocessor

  - Highest time to market due to longer development cycles associated with HW/SW modelling, hardware development, verification and testing, and integration. Highest verification overhead is present here given that one does not have the luxury of assuming the hardware accelerator has been as rigorously tested as a general purpose processor.
  - Potentially lowest power consumption if tasks are partitioned appropriately. With a low powered custom hardware accelerator one can use a more underpowered general purpose processor. This reduces overall power consumption.
  - Highest cost due to increased labor overhead resulting a larger varied team of specialized developers (HW, SW, Verification), as well as more expensive tooling (ASIC front/backend) and hardware platforms for testing (FPGAs). Additionally with the increased development time the share of labor costs as a function of product R&D cost increases significantly. More of your products cost results from labor and not materials
  - Highest performance due to tailoring the platform specifically for the application which offers greater opportunities for optimization eg. fine grain parallelism without general purpose gpus

- A general-purpose processor and a specialized processor (DSP)

  - Medium time to market. Verification overhead still higher than just a general purpose processor and this increases development time
  - Medium power consumption. DSPs still incur a power overhead during computation as well as power consumed due to communication and switching activity on busses between general purpose processor and the DSPs. The presence of DSPs will allow one to use a slightly more underpowered processor but generally the gain from a lower power processor may be overshadowed by the presence of one or more DSPs
  - Medium Cost. While general purpose processors with Dsps are popular and hence cheaper than the custom core solution the increased labor overhead for DSP programming, verification and integration will be higher than the general purpose option
  - Medium performance. Performance will be higher than general purpose option and lower than general purpose + custom accelerator. DSP usage can increase parallelism within a design but they do not offer as much fine grain optimization as the custom HW accelerator option thus limiting the avenues for optimziation.

(b) Try to sketch a potential simple strategy for exploring the design space for a given application under a given set of constraints/requirements.


create objective functions for target optimizations and their constraints
Generate initial design
evaluate design
log objective function results
modify design algorithmically (ex. genetic)
Max number of design iterations exceeded?
No
yes
create Pareto front from logged designs

Problem 1.2

(a) What is nondeterminism?

According to [1], non-determinism is described as an inconsistency in a model where for certain inputs, the behavior of the system is undefined and thus no specific outputs are guaranteed. If a model is deterministic, the same inputs will result in the same outputs. Non-determinism is different from random behavior because random behavior has a probability distribution connecting certain inputs to certain outputs. Non-deterministic models offer no guaranteed regarding outputs for certain inputs as the behavior of the system under said inputs is completely undefined.

(b) How might nondeterminism arise? (give one example not discussed in class)

Exclude :

- Statement evaluation in imperative languages:f(a++, a++)
- Concurrent process race conditions

Setup and hold time violations in Flip Flops result in the output of the saved input being non-deterministic

(c) What are the advantages and disadvantages of having nondeterminism in a language or model, i.e. in what circumstances might it be positive/desired or negative/undesired?

Positive:
When modeling uncertainty in behaviour and evaluating reliability. If I’m modeling behavior of hardware under constraints that fall outside the bounds of system specification (for example excess clock skew), I might want to evaluate how the system will fail and the severity of that failure prior to implementation and if there are any recovery opportunities. If the modeling language I’m using to model the hardware does not allow for non-determinism I will not be able to simulate the hardware under extreme conditions while considering the varied ways in which the hardware might fail. This is an example of how Non-Determinism in languages increases expressiveness and allows the modeling of more varied behaviors.

Negative:
When I am trying to express concurrent behavior without explicit synchronization, data races may occur. This will lead to non-deterministic behavior if the language has no method of resolving it dynamically, which leads to the explicit need for the programmer to use synchronization mechanisms. Unfortunately this adds programming overhead and is an example of Determinism in languages becomes a desirable quality because it limits the types of expressible behaviors and developer overhead.

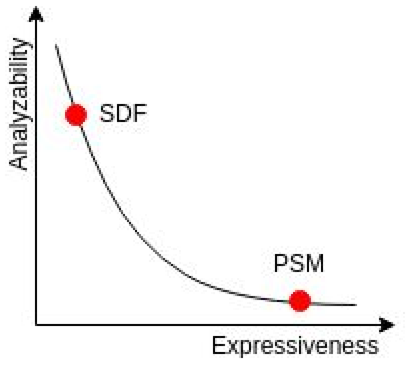
Problem 1.3

In class, we learned about the different Models of Computation (MoCs): KPN, SDF, FSM(D), HCFSM, PSM.
(a) What is the relationship between MoCs and languages?

MoC’s are models of computation, they are an abstraction of computation that allows us to reason about it. Languages are a representation of Models in machine readable forms. Example, C++ is a language that represents an imperative model of computation.

(b) Compare SDF and PSM in terms of expressiveness (how much can be captured) and analyzability. Give a use case when you would use each MoC respectively.

Given their greater generality PSMs are significantly more expressive than SDFs and they can be used to describe more varied systems. This comes at a cost of being able to reason about the system with a sufficient degree of accuracy and thus PSMs are less analyzable. We can ask more questions with PSMs but we get less definitive answers when compared to SDFs.



A use case for SDFs would scheduling and buffer sizing for a dataflow graph representing a convolution operation in a larger neural network application. SDFs will allow us to determine whether the graph is consistent, what the firing rate per agents are in a cycle and the buffer sizes for tokens being passed.

A use case for PSMs would be modeling an entire accelerator plus general purpose CPU for a neural network accelerator.

Problem 1.4

Synchronous Dataflow (SDF) (12 points) For each of the following (C)SDF graphs, indicate whether or not the graph is consistent and has a valid periodic schedule. Show the balance equations, the repetition vector, and give a minimal static schedule if one exists.

a) Answer:

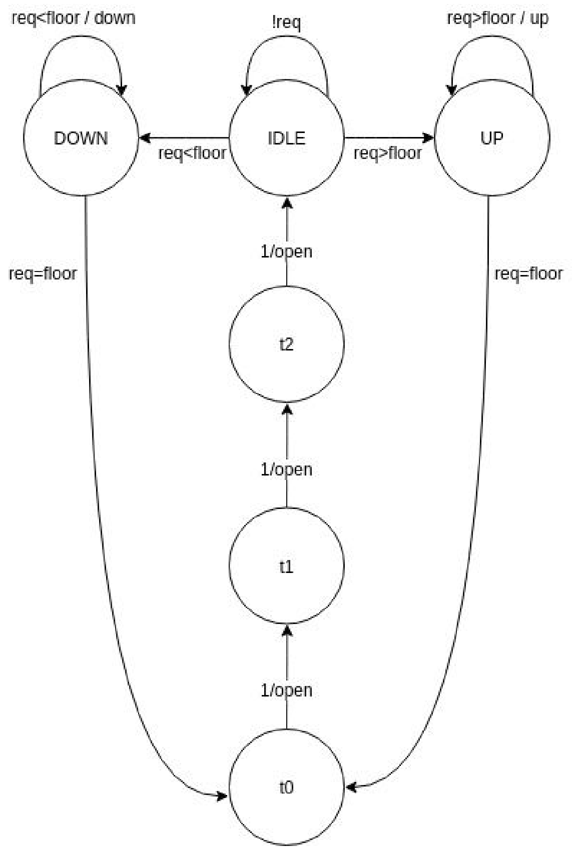
The graph is consistent provided that an initial token is present from d to c. (It can also be present from c to d but this changes the minimal static schedule presented)
Balance equation: 3a=6c=6d=4b
Repetition vector: a=4 b=3 c=2 d=2
Minimal Static Schedule for initial token from d to c: aacbbdaacbd
c) For what integer values of m and n is this graph consistent and schedulable? What is the repetition vector and a minimal static schedule when n is fixed to its minimal value and m=2n?

Integer values for m and n:

Balance equation: 4na=4nb=2nc=nd=2d
Therefore nd=4na and 2d=4na
Therefore d=4a and d=2na so n must be 2 and m >=2 to avoid deadlock
For m = 4 and n=2, repetition vector: a=1 b=1 d=4 c=2, minimal static schedule: abccdddd

Problem 1.5

State Machines (8 points) Convert the following HCFSMD model of an elevator controller consisting of two concurrent, communicating state machines into an equivalent single, flat Mealy FSM without data (i.e. an FSM, not an FSMD). The HCFSMD has two external inputs (requested and current floor), three external outputs (up/down motor control and door open signals), and two internal signals (start timer and timer timeout). Unless specified otherwise, default value for all signals is absent/0.



Problem 1.7

Program Log:

EVENT TEST

```
Sender Starting...
Sending 'H'
Receiver Starting...
Receiving 'H'
```

```
Sending 'E'
Receiving 'E'
Sending 'L'
Receiving 'L'
Sending 'L'
Receiving 'L'
Sending 'O'
Receiving 'O'
Sending ' '
Receiving ' '
Sending 'W'
Receiving 'W'
Sending 'O'
Receiving 'O'
Sending 'R'
Receiving 'R'
Sending 'L'
Receiving 'L'
Sending 'D'
Receiving 'D'
Exiting.
```

CHANNEL TEST

```
Sender Starting...
Sending 'H'
Receiver Starting...
Receiving 'H'
Sending 'E'
Receiving 'E'
Sending 'L'
Receiving 'L'
Sending 'L'
Receiving 'L'
Sending 'O'
Receiving 'O'
Sending ' '
Receiving ' '
Sending 'W'
```

```
Receiving 'W'
Sending 'O'
Receiving 'O'
Sending 'R'
Receiving 'R'
Sending 'L'
Receiving 'L'
Sending 'D'
Receiving 'D'
Exiting.
```

CHANNEL TEST NO ACK R first

```
Receiver Starting...
Sender Starting...
Sending 'H'
Sending 'e'
Sending 'l'
Sending 'l'
Sending 'o'
Sending ' '
Sending 'W'
Sending 'o'
Sending 'r'
Sending 'l'
Sending 'd'
Exiting.
```

C) Modify your code from (b) to remove all ‘Ack’ related functionality from your custom channel, compile the code, and observe its behavior in the debugger. Explain the behavior of the modified code. Is the ‘Ack’ necessary? Why or why not?

After experimenting with GDB it can be concluded that SpecC executes all parallel behaviors within a single thread without relying on any external threading library or fork commands. According to [2], no assumptions can be made regarding the order of execution for all behaviors. Additionally, as specified in [3], notify events are collected from all active behaviors until no active behaviors are left. When that occurs the notify events are sent to the waiting behaviors to be released. This behavior is present when running the code without Acks. The receiver is instantly parked and the sender thread sends and keeps raising notify events. Since the sender is the only active process available, according to [3] it will remain active until it terminates and the notifies are sent to the parked behaviors. The last notify gets received by the receiver and it is the null terminating character. This character ends the the receiver behavior and with no active behaviors the simulation terminates.

Acks are necessary to provide a synchronization mechanism between sender and receiver behavior because as stated in [2], no assumptions can be made regarding scheduling methodology. Additionally, Acks are necessary to prevent starvation and deadlocking as is the case with question 1.7c where the sender starves the receiver.

References

[1] D. Gajski, S. Abdi, A. Gerstlauer and G. Schirner, *Embedded System Design*. Dordrecht: Springer, 2009, p. 53.
[2]R. Domer, A. Gerstlauer and D. Gajski, *Cecs.uci.edu*, 2002, p. 54. [Online]. Available: http://www.cecs.uci.edu/~specc/SpecC_LRM_20.pdf. [Accessed: 01- Oct- 2019].
[3]R. Domer, A. Gerstlauer and D. Gajski, *Cecs.uci.edu*, 2002, p. 75. [Online]. Available: http://www.cecs.uci.edu/~specc/SpecC_LRM_20.pdf. [Accessed: 01- Oct- 2019].