Study of Simulink MOC in Metropolis
Haiyang Zheng

Background

Simulink is the most widely used specification capture tool for embedded control. It allows designers to model discrete and continuous time controllers and has a stateflow editing facility.

However, because the modeling environment of Simulink is a mixed and heterogeneous one as the combinations of discrete semantics, continuous semantics, etc. Usually it is not very easy to understand a Simulink model. A better way to represent a Simulink model may be necessary. One possible method is to separate the model into sub-models which have relatively simpler, well-defined semantics. With the easier understanding the sub-models, the whole model becomes more intuitive and easier to understand.

Metropolis meta-model is a communication based platform. It has a general semantics which can be customized into more concrete semantics. Thus, we try to use the meta-model to study the semantics of Simulink.

Project Goal

1. Define the semantics of a reasonable stable subset of Simulink in Metropolis, using the meta-model facilities (await, communication media, constraints, etc.)
2. Create an export script in Simulink to generate an appropriate MMM netlist using the above defined semantics.

Approach and work

A non-trivial simulink model is usually a heterogeneous model consists of both discrete and continuous semantics. More precisely, it has a discrete (periodical time triggered) model of computation as the top level, and a continuous model of computation embedded inside. A pure continuous model could be regarded as a model with the discrete period as infinity.

So, our starting point is to realize a discrete (time triggered) semantics as a refinement of the general communication semantics. The idea is to implement a timer (either as a schedule or process) which distributes the periodical events. The blocks of simulink model are implemented as processes "awaiting" for the events.
Environment Modeling in Quasi-Static Scheduling
Donald Chai

Background
QSS is a technique to reduce overhead caused by communication and context switches for a network of processes. It takes as input a set of communicating processes and generates a single sequential program that reacts to various inputs by interleaving operations of the original processes. It does this by modeling the schedule as a game between the system and its environment.

To be able to schedule more interesting processes, the current assumption is that some inputs are fully controlled by the system (the system always wins the game). However, this assumption may be too strong and lead to deadlock in the actual composition of the system with its environment.

Project Goal
Investigate conditions for the successful operation of a schedule in an environment.

Approach:
There are (at least) three approaches. They are:
- model the environment explicitly by extracting the necessary information from the environment.
- form a set of rules describing a subclass of Petri nets which are always safe to perform QSS scheduling on, but weak enough to allow interesting programs to be written.

Work
I have formulated a method in which to associate fireability of input transitions to marking of the Petri net or nodes in the reachability space. This technique is powerful, but not very efficient, and will look into the two approaches above.

Applying Ulysses to Bluetooth.
Alvise Bonivento
Mentor: Marco Sgroi

Background
In the embedded systems design the problem of IP reuse and communications between different components is becoming more and more a primary concern. Traditional design approaches to select network resources are rather informal especially for the communication domain, therefore resulting in incorrect or inefficient solutions. For this reason methodologies and tools for system design based on formal MoCs have been recently proposed. A limitation of these approaches is that they impose an early partitioning of the system behavior and many errors may occur during this phase
especially when the system operates under many but correlated scenarios. To avoid this problem it is necessary to switch to a design paradigm that is based on the following flow:

1. The designer specify separately all the scenarios that describe the behavior of the system.
2. Tools derive an equivalent executable model from which an hw/sw implementation can be synthesized.

Ulysses is a new approach to protocol design based on synthesis from a scenario-based specification. Ulysses is based on the following principles:

- Separation of communication and computation: the interaction among protocol entities is specified independently from data processing protocol functions.
- Separation of function and architecture.
- Protocols scenarios are specified using sets of related MSCs, called Message Sequence Nets (MSN), that are PNs whose transition are labeled with basic MSCs. The PN model allows the representation of concurrency and provides a formal model to check relevant properties like deadlock and race conditions.
- Use of design patterns to simplify the design process. Often happens in protocols design to have the same function or procedure repeated many times, it is therefore useful capture this aspects in special patterns. Rules of composition are then provided in order to guaranteed consistency and correct by construction design.
- Tradeoff architectures to minimize the implementation cost.

**Project Goal and Approach**

The project consists of applying the Ulysses methodology to the Bluetooth protocol. This will require describing the Bluetooth protocol in MSCs, formalizing the relevant patterns and deriving the consistent PNs models. The Bluetooth protocol suits particularly well for the test of this new methodology since it offers an overall structure which is different from the usual implementations of the OSI stack layer, offering parallel sub-layers and interactions between non adjacent layers. In such a singular approach (also due to the necessity of optimization for power concerns) the application of a formal design methodology like Ulysses can provide significant benefits. Furthermore the increasing need of communications between devices that can be very different between each other will eventually result in protocol solutions that will offer aspects more and more distant from the classical OSI model and Bluetooth is just a step into this direction.

Like all the new tools, Ulysses needs to be developed since there are some particular aspects of the protocol for which specification via MSNs do not seem to be the most natural approach. Another interesting dimension to be explored is the idea of building a fractal model for patterns (each pattern is the composition of other sub-patterns). It is actually our hope to encounter as many problems as possible on the way and to find consistent solutions for them in order to enhance the model and provide a wider range on patterns and eventually creating a pseudo-library of PNs models of the most common functions.
Where did we get so far?
We already developed an almost complete MSNs model for the Link Manager, the Host Control Interface and evaluated solutions for most of the remaining functionalities of the protocol. We are also on the way to start deriving the PNs model for a complete communication session at least for the layers above the Link Controller.

Low Power Network Discovery Algorithms
Abhijit Davare
David Nguyen

Background
An ad-hoc sensor network is a network of numerous wireless nodes where each node has sensing, storage, computation, communication and possibly actuating resources. Each node has a limited power source that determines the lifetime of the node. Therefore, energy is the most critical factor in sensor networks in all of its design levels (i.e. hardware, network layers and applications). Numerous studies in sensor networks have already demonstrated that communication between nodes is the dominating source of energy consumption. While traditional wireless network architecture has been based on systems of static base stations, it appears that a multi-hop network, where each node communicates with a few close nodes, is most efficient in terms of energy savings and bandwidth reuse. In multi-hop networks, each node communicates with other geographically distant nodes using intermediate nodes to build a communication path. In order for sensor network tasks to be scalable, all of the algorithms should be localized in the sense that each node performs its tasks based only on its local information.

This research introduces and evaluates new power saving techniques for wireless ad-hoc sensor network. We first develop efficient and localized coordination protocols to schedule the activities of the nodes in the network and put the idle nodes in the standby mode. We also address the problem of finding an efficient and localized information gathering algorithm that minimizes the total number of radio communications during the task of gathering complete information from all of the nodes within a network or a specified part/subset of a network. The assumption is that each node has all the information about itself (including its geographic coordinates and its data) and all of the information from the nodes within its communication range.

In order to present the flow of the algorithm within the network, we introduce the notion of \( \text{it token}\). \( \text{it Token(s)}\) defines a node that is currently active and is executing the algorithm. The procedure involves electing the \( \text{it next active node}\) and giving the control of the algorithm (token) to that node. The token node can only send control to the nodes within its communication range (neighboring node). This algorithm can utilize single or multiple tokens. We start by demonstrating our approach using a single token and then show how we can generalize the algorithm to use multiple tokens.

Project Goals
Since many nodes in the network can be idle, we can save power by turning the radios of these nodes off and putting them in standby mode. In the first part of the project, we describe and evaluate a localized technique for reducing the energy consumption of the
network by putting the maximum number of nodes in standby mode without affecting the connectivity of the network, \{Xu01,Che01\}. Each node determines its status based on its available energy and the connectivity of the network within its neighborhood. We have developed necessary and sufficient conditions for the local procedure for checking the connectivity of the network. Interestingly, the same mechanism is the basis for efficient power management techniques. Our power management technique finds the provably minimum power state of the nodes within a local neighborhood without affecting the connectivity of the nodes, while a subset of the nodes are staying active and are communicating.

Each node that changes its status to standby mode broadcasts its status to its neighbors before going to sleep. The second part of the project is the design and evaluation of a low power localized procedure for gathering information from the active nodes in a network while some nodes are in standby mode. By information, we mean a type of meta-data that can be different for different applications. In the simplest case, this information is just the status (active/standby) of all the nodes within the desired area. In order to find the information contained in a node, either the node or one of its neighbors should be visited. This type of data gathering is NP-complete. Since the data gathering procedure is localized and the network is ad-hoc, at each active node the important part of the procedure is to decide where next to send the token. We provide an efficient, yet complete, low-power heuristic for data gathering. By complete, we mean that we have, provably, all the required information at the termination of the algorithm.

**Approach**

The procedure for assigning nodes to standby works as follows: a local decision is made at each node based on the alternative routes between its neighbors and the amount of energy it has. The necessary and sufficient condition for a node to go to sleep without changing the connectivity of the network is that an alternative path must exist between each pair of its neighbors. A perfect sleeping strategy should keep the length of the ad hoc routes of the network at the minimum state, as if the network was fully connected. Therefore, if a node’s inactivity would cause the alternative paths between its neighbors to be much longer, the node would not go to standby mode. In order to have a fair balance between the activities of the nodes, nodes in standby periodically wake up and check if they should change their status with the currently active nodes that change the energy level. The approach is tested against another recently proposed coordination strategy, called SPIN \{Che01\}.

The information gathering procedure is built on the observation that different network shapes dictate different traversal strategies. The procedure starts by exploring the shape of the network, using a geometric perimeter routing (outer face routing) technique \{Bos99\}. The nodes on the perimeter are assigned a rank of zero, while all of the other nodes in the network rank themselves according to their minimum distance from the perimeter nodes. The next active node is selected based on its rank, number of neighbors whose status is unknown, number second level neighbors whose status is unknown and the area constraints. The area constraint is the component of the algorithm that ensures all nodes of an area are visited before leaving that area. Otherwise, the procedure will encounter the additional energy overhead of visiting redundant nodes to return to unvisited nodes from an already visited area. We illustrate the important of articulation
points in this area based approach. Articulation points partition the graph into a number of subgraphs. The procedure uses the bounds of each of these subgraphs as the area constraints. We illustrate how tuning the different parameters of the procedure can yield a more stable method for traversing the graph. Also, we study the performance of the algorithm for different shapes of the network. We evaluate the performance of the algorithm against instances with a known optimal solution to study the efficiency of this approach.

We calculate the total power consumption in terms of the number of transmission and receptions in the network. If the energy cost of a transmission is A and the energy cost of a reception is B, the total energy cost would then become: A* (Number of transmissions) + B* (Number of receptions).

**Status of the Project**

So far, we have formulated both of the target problems that we are working on. We have also discussed and outlined the pseudocode for both of the problems. The simulator for the localized network topology coordination procedure and the localized information gathering procedure is written in C++. We have simulated the network by uniform random placement of the nodes and using a unit graph model for the nodes. In terms of implementation, a lot of progress has been made for the implementation of the second part (i.e. information gathering). The perimeter routing and relaxations to find the rank of the nodes (i.e. shortest distance of the node to a perimeter node) are already implemented. We identify the perimeter nodes and divide the space into subregions accordingly. We have also used the three components (i.e. energy, unvisited neighbors, and area) to find the next active node and go through the nodes in the network. Currently, we are implementing the generator of instances with known optimal solution to test the performance of the algorithm against it. We will also study the performance of the algorithms for different network shapes. We are also starting to implement the localized energy efficient coordination procedure. The output of this procedure would serve as the input for the information gathering portion.


**Modeling continuous-time sub-systems in Metropolis**

**Gabriel Eirea**

**Background**

Metropolis is intended as a design environment for heterogeneous embedded systems. In
particular, for many applications it is very useful to model a continuous-time (CT) sub-

system using the same representation model as the rest of the system. This would enable
not only simulation, but also the use of analysis and synthesis tools over the whole

system.

Following, we identify some scenarios in which the modeling of CT sub-systems would

be valuable:

1) Sampled-data systems. The most common form of interaction between the discrete and

the CT domain arises from the use of computers to control some physical process. In

most cases, this is achieved by sampling the continuous variables and writing control

actions to a zero-order hold circuit at discrete time instants, usually in a periodic fashion.

2) Hybrid systems. Some CT system can't be described by ODEs, because the dynamics

change abruptly when the state variables reach certain values (e.g., a collision), or when

an external command is applied (e.g., a switch relay is activated). Therefore, they are

modeled as a finite state machine, in which each state represents the evolution of the

system as a set of ODEs, but transitions between discrete states may occur.

3) Event driven systems. In some cases, the system should be able to detect when a

certain condition is satisfied by the CT variables, in order to execute certain tasks (e.g.,
temperature greater than 25 degrees). An event is generated as a consequence, and

processed by the discrete system. While in hybrid systems, these events could indicate a

change in the CT dynamics, in event driven systems, the events are used simply as

information to be processed, and don't have necessarily an effect on the dynamics.

Goal

The goal of the project is to develop a general methodology for modeling CT sub-systems

using the Metropolis meta-model. The CT behavior is described by ordinary differential

equations (ODE), where time is the independent variable. An interface between a CT

block and the rest of the system will be defined, and several modeling examples will be

generated.

The main challenge in this project is how to deal with the notion of time. While this is an

essential notion in CT systems, in many models of computation it is not explicitly

included, or the semantics refer only to the concepts of causality or ordering of events.

Approach

First, we will try to identify the most efficient way of representing the interface between

the CT sub-system and the rest of the system, and define a medium that implements this

interface. In principle, we will assume that whenever a value is written into the CT

inputs, a zero-order hold will retain this value until the same input is written again. A

problem to be addressed is how to determine the validity of an input when the CT process

reads it. Moreover, the semantics of the interaction have to specify when the execution of

the process thread is enabled. Two approaches could be used: the CT process is executed

when an input is provided, or when an output is required. In general, the timing of inputs

and outputs should not be determined by the CT process, but the rest of the system (either
the medium or other processes), unless there is an event detected, in which case the CT process should indicate it thru the medium. In some cases, the event to be detected is an integral part of the CT model, but in other cases it is not, and a mechanism should make it possible to indicate the CT process which events are to be detected. We will explore the use of the GlobalTime quantity and its role in this interaction.

Next, we will study how to embed an ODE integrator into a process, trying to make this process as general as possible, so we could use different ODE solvers with minimum changes in the code. If time permits, we will study how to make the ODE solver detect events and communicate them to the rest of the system thru the interface.

The final outcome at the completion of this project, is expected to be a formal definition of the interface, and its implementation as a medium object, plus a (probably simple) ODE solver coded into a process. Both together would conform a "template" that could be used as a reference for future models. Several examples will be developed to show the use of this template, and simulations will be carried out to demonstrate its performance.

Work

So far, we have been working on ideas for the definition of the interface between the CT sub-system and the rest of the system. We are concentrating our efforts initially on sampled-data systems.

The semantics of the interface is defined as follows:
1) an actuator writes to the CT sub-system a triplet (value, time, expiration time)
2) a sensor requests a read from the CT sub-system indicating the time of the desired output
3) if the expiration time of all inputs are greater than the read time, then the CT sub-system integrates the ODE from the previous state until the time of sensor request; otherwise the read is blocked until all inputs are available
4) an "ordering" condition is to be imposed to the system events, such that no sensor read can be requested at a time before than the write time of any the actuators; this will make it possible to discard old input values, such that no queue is needed in the interface. We believe that this interface semantics enables the modeling of sampled-data systems, even in the case when different components interact with the CT sub-system at different sampling rates. Moreover, the sampling times don't need to be periodic, as long as the expiration time of each value is known.

Metropolis SystemC Based Simulator with the support of Mapped Behavior and Performance Indices
Guang Yang
Daniele Gasperini
Mentor: Yoshi Watanabe
Felice Balarin
To handle the increasing design complexity, platform-based design has been proposed. In this design methodology, a design can be viewed as two parts, the supporting platform and the behavior sitting on top of the platform. The platform provides some software and hardware services. It also provides performance estimation for these services. The lower level a platform resides, the more accurate performance estimation it can provide. The other part, behavior, defines pure functionality of the system. It has no information about how these behavior will be implemented, thus no knowledge about the performance of them. Then, designers can decide how the behavior would be implemented. This corresponds to choose or develop a particular platform. Typically, to speed up the design process, designers often start from system level platforms. And then, successively refine the platforms down to real HW/SW implementations. One possible trace of such refinements could be transaction level, RTL level, gate level, circuit level and finally real HW/SW level. No matter on which level a platform is, after mapping behavior to that platform (i.e. specify which piece of functionality is implemented by which service(s) provided by the platform), it is necessary to evaluate how good the behavior is performed on that platform under the mapping. This evaluation can be used to optimize the platform, the behavior and the mapping. This project is targeting such an evaluation tool.

Currently, we are doing our work in Metropolis, a research project backed by the concept of platform based design. Our goal is to extend the Metropolis SystemC based simulation tool, and make the performance evaluation possible. In the new simulator, behavior and platform (or architecture) will be co-simulated. By using special constructs provided by Metropolis MetaModel (mmm, design language in Metropolis) defined in both behavior and platform, behavior and platform can be tightly synchronized. At each synchronization point, the behavior part can ask the performance indices from platform.

In this approach, we do not take either hardware centric or software centric co-simulation techniques. We simulate behavior and platform parts as they are. Since we do not require software model of hardware or hardware-like model of software, we could expect a high simulation speed. And all designer’s energy can be spent on modeling both parts as efficient as possible and leave all simulation concerns to the simulator.

In order to achieve the above approach, we must incorporate following functions into the simulator.

1. Define two networks: the user-provided real network (including both behavior and architecture parts) and the automatically generated or user-provided scheduling network (corresponding to both parts) (Please refer to the technical details for detailed scheduling network structure.)
2. Register synchronization points from both parts to simulation manager
3. Register performance indices’ requests to simulation manager
4. When coming to a synchronization point, resolve performance indices and reach a fix-point for all indices in the platform
5. Pass performance indices from platform to behavior part
6. Coordinate simulation to let it run one synchronized step each time
So far, we have investigated the simulation strategy and proposed where to perform above functions. For each function, we devised sketchy algorithms and defined some basic supporting classes. The rest of our work would mainly be implementation. Also, there is a lot of testing work.

**Mapped Behavior**

1. **Actions**
   In mmm, we define actions as the executions of function calls, a single statement and even an expression or a sub-expression. During simulation, each process generates a sequence of actions. If we have multiple processes, the simulation is composed of a sequence of action vectors.

   ```
   class action {
   public:
   process *p;
   sc_string label;
       process * getProcess();
       void setProcess(process * pr);
       sc_string getLabel();
       void setLabel(sc_string l);
   }
   ```

2. **Events**
   Associated with each action, there are two events, the beginning and the end of that action. Event is the critical concept in this project. The synchronization between behavior and architecture sides is through events. Also, when we want to get back the performance number from architecture side, we annotate the number with an event in behavioral side.

   ```
   class event {
   public:
   process * p;
   action * a;
   process* getProcess();
   void setProcess(process * pr);
   action * getAction();
   void setAction(action * ac);
   }
3. Register events to simulation manager and quantity managers
This is done by calling the service function defined in scoreboard, which is a part of simulation manager. In scoreboard, there is a list of processes. For each process, the scoreboard records every single movement for it, including action and event. This piece of information could be later on used by simulation manager to do synchronization and scheduling.

Add the following functions into scoreboard.
void registerAction(process* p, sc_string l);
void registerEvent(action* a, bool begin_or_end);
void registerEvent(process* p, sc_string l, bool begin_or_end);
action* getAction(process* p);
event* getEvent(action* a);
event* getEvent(process* p);

4. Resolve quantities and do scheduling
This part is tightly coupled with Quantity Manager stated below.

Quantity Managers
1. Introduction
Two networks are defined: the real network and the scheduling network.
In the following sections we are dealing with the scheduling network.
A good part of it is predefined, but the user may also extend it, as described below.
In every state of function netlist, the scheduling netlist is "executed" to determine what happens next.
The "execution" is defined precisely below, after describing the network.
The terms network and netlist are used in this context interchangeably.

2. Scheduling network structure
The scheduling network consists of the following objects:
   • one scheduler for each netlist
   • one state-medium for each process and medium
   • one manager for each quantity in the scope
   • any additional, user-defined, processes and media

A quantity is in the scope if it is defined in the netlist or it is passed as argument in the netlist constructor.
As stated above a good part of the scheduling netlist is predefined.

- In particular default connections in the scheduling networks involve:
  - each statemedium, that has one port for each quantity in scope (named the same as the quantity);
  - each netlist scheduler that has one port for each statemedium in the netlist (named the same as the process or medium), one port for each sub-netlist scheduler (named the same as the sub-netlist) and one port for each quantity manager (named the same as the quantity);
  - each QM that has one port for each statemedium in whose scope it is.

In other words if quantity manager QM is in scope of statemedium SM, then QM has a port named SM (to make calls such as setMustDo and getCanDo) and SM has a port named QM (to make requests).

3. Scheduling network objects
The proposed interfaces of the objects in the scheduling network are: quantity managers, state media (sm) and netlist schedulers.

4. Executing the scheduling network
At each state of the scheduled network, a scheduling network is executed in two stages:
1. request making phase
2. resolution phase

In the request making phase, each thread executes the "request making code (RM code)" associated with the current state. RM code is code enclosed by \{ $ \}$ in the metamodel language.

There exist some open topics. For example the results of executing RM codes of different processes could be ambiguous (are they executing concurrently? in order? which order?...)

The resolution phase proceeds as follows:
- the function ifTop() is called for the top-level netlist
- this may un-block some user-defined processes in the scheduling netlist
- the execution continues until ifTop returns, and all user-defined processes in the scheduling netlist become blocked again
- we require that during execution the function setMustDo of every statemedium is called exactly once, in other words, finishing the execution must imply that resolved at the top level is true

There is only one request method, however, each request is packaged into a class inheriting from requestClass which also carry canBeDenied flag.
The canBeDenied() method can be defined by the user for each refinement of requestClass, i.e. all requests of the same class will have the same flag.

5. Scheduling network defaults
In this section we describe default implementation of functions of scheduling network objects. If a function is not mentioned here, it means that the user may not redefine it.
The code presented is pseudo-code.

**Quantity managers**

```c
void request(event e, request r) {}

void resolve() {
    for each statmedium sm in scope {
        // just pick one
        if(!sm.resolved())
            sm.setMustDo(sm.getCanDo()[0];
    }
}

void postcond() {}
```

**Statemedia**

```c
void postcond() {}
```

**Netlist schedulers**

```c
RM code none

void resolve() {
    if(!resolved()) {
        call resolve() for all QM’s defined in the current netlist
        call resolve() for all sub-netlists
    }
}

void postcond() {
    call postcond() for all QM’s defined in the current netlist
    call postcond() for statemedia and processes
    call postcond() for all sub-netlists
}
```

Note that all the calls above can be made through pre-defined ports of the calling objects. This way we maintain communicate-through-ports-only paradigm

```c
void ifTop() {
    while(!stable()) {
        resolve();
    }

   forall( statemedia sm in the whole system) {
        // just pick one
        if(!sm.resolved())
```
This method is called for the top netlist and triggers the annotation of events of all the requested quantities and the scheduling of the events that can be executed.

**A Methodology for the Computation of an Upper Bound on Noise Current Spectrum of CMOS Switching Activity**

Haibo Zeng (zenghb@eecs), Joshua Garrett (joshuag@eecs)

In mixed signal designs, the switching activity of CMOS digital blocks is responsible for noise currents injected into the power supply system and substrate. These injected currents may affect the reliability and performance of sensitive analog components. To verify correct operation, we would like to quantify this noise. For practical reasons, a measure of the noise current cannot be done through exhaustive simulation; rather, we would like to determine an upper bound for the spectrum of the noise current with respect to all possible transitions of circuit inputs.

The project goal is to verify and improve the accuracy and utility of the upper bound current spectrum algorithm in [1], as demonstrated through several benchmark digital circuits. In order to implement the algorithm, we will have characterized a set of CMOS standard cells with respect to the input timing and output load dependence of noise current. This approach is novel in that it does not require exhaustive simulation of the circuit under test (in fact, the algorithm complexity is shown to be only linear in the number of gates), and yields an upper bound in the frequency domain, which is the most useful point of reference for analog and rf circuit designers who wish to ensure robust operation of their sensitive components.

The algorithm for computing the upper bound of the noise current spectrum was proposed by Dr. Alessandra Nardi in [1]. This algorithm will be tested in this project, and improvements made where possible to increase accuracy or reduce computational complexity. To test the algorithm, we will need to characterize a subset of gates from the ST 0.13? m process library, producing look-up tables for each of the $2^{2^p}$ input transitions
(for a gate with $p$ inputs) which relate noise current values to output load capacitance and input transition times. The form of these look-up tables will have a one-to-one correspondence with those provided for delay calculation, since these templates represent the useful range of input timing versus output load for each cell in the library. Several benchmarks will be used for test cases, in order to compare the upper bound generated from the algorithm to that measured by full simulation. These benchmarks will be converted from native BLIF (Berkeley Interchange Format) format to VHDL, synthesized into gate level Spectre netlists using Synopsys Design Compiler, and imported into Cadence. These netlists will also be imported into SIS, so that the graph traversal and logic simulation necessary for the algorithm can be performed.

Before exploring the algorithm, we need to verify the accuracy of the gate current spectrum model itself by comparing the current injection and the corresponding spectrum according to a circuit simulation and according a reconstruction from the characterized library. We need to choose the input transition times and arrival times of the primary inputs in order to generate on the internal nodes all the particularly critical cases.

To date, Dr. Nardi has outlined much of the algorithm and characterization process, but scripts for cell characterization (targeted for an older 0.18?m process technology) and implementing the algorithm remain unfinished. Our work to this point has been in studying the previous work in order to understand how to prove the proposed algorithm. In order to facilitate the characterization process, some preliminary tests of the simulation tools and a new process technology have been done.


**Implementing Run-Time Support for DRAFTS**

**Mark McKelvin**

**Background:**

The focus of this project is to implement a run-time support environment for the Distributed Real-time Applications Fault Tolerance Scheduling (DRAFTS) project. DRAFTS is a project that addresses the issue of automating the synthesis process from a
fault tolerant data-flow (FTDF) model, that is based on the underlying classic data-flow model of computation, to executable code such that the implementation is fault-tolerant and meets real-time constraints. Unlike the classic data-flow model, the FTDF model has firing rules that allow an actor to fire when a subset of inputs is missing. It is based on assumptions on the distributed run-time environment that is characterized by its ability to perform message handling across distributed processes, static scheduling, and coordination between processors. Furthermore, the run-time environment will be used as a performance model to provide timing information about the FTDF. This will allow the designer to address the issue of scheduling computation and communication to meet real-time constraints.

**Project Goals**

The overall goal of this project is to implement a performance model that can be used by a designer to implement a FTDF model. The environment is to include a library of functions that may be used by the user or designer to implement a FTDF network. Primarily, my work is to assure the scheduling of computation and communication for a predefined set of actors that must operate according to the FTDF model specifications. One step to accomplishing the above goal is to model the communication of actors that may be placed on different nodes, or processors, and to model the communication of actors on the same processor. Keeping in mind platform-based design concepts and modularity, the communication scheme between actors used should ideally be able to be implemented on various architectures. Given that a library for specifying and simulating FTDF actors in Metropolis exists, the same interfaces between actors and communication media should be implemented. The final step is to implement a static scheduler that will schedule the communication and computation between actors.

**Approach**

The general approach taken to accomplish the aforementioned set of goals is simply a communication-based approach with the concepts of platform-based design at hand. The actual implementation is based on the concept of using a micro-kernel structure that is modular and can be used on multiple processors and various architectures (ideally). Due to the aspect of running a simulation and a lack of appropriate resources, a single processor computer system running Unix or one of its variations will be used to run multiple copies of the micro-kernel, each representing a physical processor in the network. In addition, the actual implementation will allow the micro-kernel and its related set of functions to run on multiple computer systems, so to fully test also communication on the physical channel.

As stated before, actors are specified using a library called the FTDF library. This specification includes information such as the number of inputs, the number of outputs, the computation, the destination and sending actors, and the firing rules for an actor. An interface between the actors and the computer network (medium) will be created. This interface will have the ability to send and receive data through a medium that connects multiple computer systems. To simulate the communication between actors on different
computer systems, the plan is to use User Datagram Protocol (UDP) sockets over the Internet Protocol (IP) to establish a physical communication channel. UDP is chosen over the Transport Control Protocol (TCP) because it is unreliable, but yet, it supports the ability to address data from source to destination computers. This is also parallel with the FTDF model since its purpose is to be able to operate and respond to an unreliable communication medium. Therefore, networked computers using UDP sockets will serve as the interface to the physical IP medium that will implement the communication between actors on different computer systems. Actors on the same computer will be implemented using a shared memory concept on a single computer that plays host to multiple communicating actors. Figure 1 below gives a very abstract example of how the interfacing (shown in red), or micro-kernel, will cooperate with the user processes developed using the FTDF library and the underlying network interface.

![Diagram of Processor 1 with layers: User Application, FTDF Library, Communication Tasks, Scheduling, Host Operating System, Hardware/Network]
Figure 1: The middle portion (displayed in red) is an interface between the FTDF library and the underlying network. The interface will offer communication tasks used to communicate between local and remote processes and a scheduler to schedule all of its tasks.

**Work**

The work done thus far has primarily been focused on creating functions that interfaces the physical medium to the incoming and outgoing data between two computer systems by using UDP sockets and IP. I have created code that act as user functions that will physically network multiple computers given an address using UDP over IP. I have also begun work on communication between multiple processors on the same processor. This requires an interface to be created that will route data from one actor to another. This interface will be composed of ports (attached directly to the actors), a driver, and a function that will have the ability to route or receive data to and from actors on the same computer or on another computer. I have created a structure that can be used to initialize ports on the actors that will facilitate the actual connection between the actor and a messaging task that will handle the routing of data from a source port on an actor to the correct destination port of an actor.

**Models of Computation for the Robot Diffusion Problem**

**Sarah Bergbreiter**

**Background**

Distributed control systems provide an interesting domain for embedded system design due to the reactive nature and complexity in timing inherent in these systems. Distributed control can be defined as a control algorithm characterized by multiple sensors, actuators, computation elements, or combinations thereof distributed through space. For example in a building HVAC system, temperature sensors and actuators controlling the heating system are distributed physically. It is natural to think that these distributed control algorithms could be implemented on small, wirelessly networked nodes similar to those built by the TinyOS group at UC Berkeley [1].

This project will examine one particular application of distributed control, robot diffusion. In the robot diffusion problem, \( N \) robots reside in a given area \( A \). The goal for these robots is to diffuse outwards to maximize their distance between each other. Robot diffusion had distributed sensing (between robots), actuation and computation making it an excellent candidate on which to study distributed control. One application for robot diffusion would be to deploy a mobile sensor network in order to cover the largest possible area with sensors [2]. A search and rescue operation would be a perfect venue for such a task. Yet another application more specific to robots is “diffusion mapping”, where line-of-sight connectivity determined while the robots diffuse outward is used to determine a map of an unknown area [3].

**Project Goal**
Naturally, as with most control problems, some of the key parameters that we are interested in proving are the stability and convergence of the algorithm. Many algorithms in the field of distributed control put communication synchronicity assumptions in the control algorithm to guarantee stability in convergence. However, due to the wireless network connecting the nodes of our system, it is likely that communication will not be synchronous at all. For example, in [4], stability of vehicle positions in a one-dimensional problem is directly tied to the communication sampling period. If this period is too large, the system will become unstable.

The goal of this project is to formalize and abstract the robot diffusion problem and to examine an algorithmic solution under various conditions of synchronicity. For example if the system is considered entirely synchronous (where all robots share the same notion of time), it is likely that several key properties such as stability will be provable. However, given increasing asynchronous behavior, I would like to determine where and how these properties fail. This will be accomplished through simulation and on real robots.

**Approach**

The first step in this project will be modeling the robot diffusion problem and a controller in the context of a reactive embedded system. Reference [4] describes the one-dimensional problem of robot diffusion as a discrete-time system. However, the goal of this project is to pay much closer attention to the time properties of the system. Therefore, an interesting test arises when modeling the system as a network of FSMs that communicate synchronously versus a network of FSMs that communicate asynchronously (the GALS – globally asynchronous, locally synchronous – philosophy).

After the controller has been modeled as a network of FSMs (where an FSM resides on each robot), I will write Esterel modules to describe each FSM. Simulating this network of modules in parallel will give a good idea of the performance for a network of robots who share the same notion of time. By modifying the simulation slightly to include various degrees of clock skew in the communication between the modules, I can model varying degrees of asynchronicity. Various stability tests will be run on the system to test performance under various timing constraints (for communication between robots).

Finally, because Esterel compiles into C, I can synthesize code from the Esterel model to run on real robots. The robots use the networked sensor nodes built by the TinyOS group at UC Berkeley [1] along with a toy RC car base to move and communicate. Due to the complexity of the localization problem however, I will only demonstrate a 1-D diffusion model in real hardware.

**Work**

The first work needed was to formalize the problem specification and to develop the controller for robot diffusion. To date, work has been concentrated on this point and for the one-dimensional case based on work done in [4] and a Simulink model has been built and its effectiveness demonstrated. In this example, the only information shared is the position of the neighboring robots.
However, an even simpler hardware implementation would require that the controller know only the distance between the robots instead of absolute position (which is often more difficult to measure in reality). Therefore, this controller has been modified to share information on distance between the robots instead of position information. This controller has also been implemented in Simulink and tested there, with the next step being to translate this into a Finite State Machine representation.

In addition to designing the controller, I have also been familiarizing myself with the Esterel environment and what will be required to translate the code from Esterel’s C to the code required by the TinyOS environment.


Effective Models for Network Application Design

Mel Tsai

Background

There are currently a wide variety of tools for network design, simulation, and application prototyping. A very popular system in the category of modular routers is MIT’s Click Modular Router language, which allows users to quickly prototype and develop network applications & protocols. By stringing together a series of pre-written network elements in the Click language, a network routing application can be built with useful functionality. Furthermore, the Click runtime environment can integrate this router application into the network stack of a Linux machine, allowing users to physically implement the network application on commodity hardware with relative ease.

Unfortunately, as with some other modular router packages, the Click Modular Router and its semantics have a number of shortcomings:

1. Click’s unique “push/pull” semantics are too restrictive for describing applications on the architectures of many embedded network devices. These semantics may
work well on a Linux PC with only one or two Ethernet ports, but they quickly become inefficient once more network ports are added, causing the size of the Click description of the application to grow in direct proportion to the number of ports in some cases.

2. The push/pull semantics of Click are not rich enough to describe many types of desired functionality (i.e. certain QoS-related policies), especially when the number of network ports is moderate to large.

3. Click’s semantics cannot adequately express concurrency in a design. This prevents the Click language from being used in other domains such as network ASIC design and the simulation/implementation of applications for large-scale routers and switches.

4. While the Click element database contains more than 200 elements, the database itself is rarely rich enough to implement protocols other than those found in standard routers. Even simple network applications can sometimes require the creation of custom elements in C++, depending on the desired functionality. For example, this is often the case for applications that use packet types other than TCP/IP packets.

**Project Goals**

The primary goal of this project is to develop a new set of semantics and integrate these into the Click Modular Router framework. These semantics should address the above shortcomings of Click, including greater descriptive power and the representation of concurrency in the design. I will keep in mind that my new semantics may have other shortcomings, and comment on these once I have evaluated the results.

Secondary goals of the project include the creation of an enhanced simulation facility for Click as well as the creation of a canonical set of Click elements (using my new semantics) that will decrease the likelihood that a user must write custom elements in C++ when implementing a new network application or protocol.

A “stretch goal” (one that I will attempt but probably will not finish) is to use my enhanced version of Click in other domains. Specifically, I may write a small compiler that translates a Click application description using the new semantics into a valid configuration script for a large-scale router. I will compare the feasibility of this translation (using my new semantics) verses standard Click. In my research I have access to a large carrier-class Nortel router (a Passport 8600) that I can use in such experiments.

**Approach**

The basic steps in this project are as follows:

- Perform a brief survey of currently-available tools for the design & simulation of network applications, with an emphasis on modular router packages such as Click.

- Study several models of computation (i.e. KPN) in order to develop a feel for which one is best-suited for the new Click semantics. Together with the background research into other modular router packages, this will form the basis for the design of my new semantics for Click.
• Once new semantics have been chosen, I will strip out the internals of the Click runtime and replace them with a new runtime environment that implements these semantics. (The Click source is freely available and I am quite familiar with it.)

• After the new semantics have been created, I will create a few elements for these semantics and implement a few example applications. These examples will highlight the differences in design entry & efficiency verses the regular semantics.

• Specify (but not necessarily implement) a canonical set of network elements.

• In order to measure results more easily, I will extend the simulation functionality of Click. In previous projects I have started to extend the Click simulation features, but I have never completed this to my satisfaction, and this project has unique simulator requirements.

• Maybe write the Click translator for the Nortel Passport 8600 series router.

Completed Work

Thus far I have read a few papers on other modular router systems. I have also begun to analyze the feasibility of implementing a new run-time environment on top of the Click environment. Using Click as a wrapper for a new system will save a significant amount of work, but it presents some challenges that need to be addressed. Concurrently with EE249 I am also implementing an advanced networking platform (several linux PCs, layer-7 routers, and two carrier-class Passport routers) that I can also use to generate results for this project once I have modified Click.

RTL Code Generation for Teepee

Nathan Kitchen
Vinay Krishnan

Background

The Teepee framework is the design environment for the Mescal project. Teepee will generate both the simulator of a processor and synthesizable RTL code from a single formal description so that the simulation description and synthesis description of the design can be verified to be equivalent.

Project Goal

The project purpose is to implement RTL code generation for Teepee and show that the RTL description of the system is equivalent to the simulator.

Approach

First milestone

We aim to generate from the abstraction of the processor a very basic Verilog circuit description. This shall functionally be equivalent to the simulator. For this milestone, we will identify the functional units that are needed for each instruction and copy them so
that there is a separate mini-datapath for each instruction. (This is the structure of the simulator.) We will select the output of a mini-datapath based on the current instruction.

**Second milestone**
The next step is (conceptually) to fold the separate datapaths back together into the original circuit topology. We will already know the necessary control signals for each functional unit from the first phase. We just need to read the right set of signals from a lookup table using the instruction tag as an index. One complication is that the processor may have multiple issue width, so we will have to find a way to combine sets of signals.

**Equivalence**
We will show that our code generation is correct by simulating the Verilog produced and comparing its outputs to the outputs of the simulator. We have many sample designs that we can use for this.

**Work**
We have familiarized ourselves with the Teepee framework. We have experimented with sample designs in the graphical. We studied the expression language and constraint language that is used to specify the datapath and control flow. We have also inspected the flow of the hardware simulator generator and understood how it simulates the hardware. On the implementation language side, we have also familiarized ourselves with the Java language and the Verilog hardware description language.

**Communication-Minimized Distributed Computation**

**Jie-Hong Jiang**

**Background and Previous Work**
As an embedded system interact with its environment, which may span over a large area, it is sometimes necessary to distribute a computation task to some particular locations. In such cases, the communication costs among different locations may be quite expensive or unreliable. Therefore, it is important to derive algorithms distributing computation tasks under physical constraints with minimal use of communication resources.

Communication complexity has been intensively studied in the community of theoretical computer science for over two decades. Previous work focused on proving lower and upper bounds of communication complexity, assuming computation models have unbounded computation power. However, this assumption usually violates realistic situations, especially for embedded systems. As a result, communication complexity in a design perspective is relatively new in the literature. Here we study communication complexity in the context of embedded systems.

**Objective**
Given a computation task and a geometric partition of its inputs and outputs, we study the limit of least communication among different blocks induced by the partition.
In particular, we consider the communication between two parties who have bounded computation power subject to implementation constraints.

**Problem Formulation**

In this project we explore the solution to the problem formulated as follows. Let $C(I,O)$ be a computation task (either combinational or sequential) with sets $I$ and $O$ as inputs and outputs respectively. Suppose $I = I_1 \cup I_2$ (with $I_1 \cap I_2 = \emptyset$) and $O = O_1 \cup O_2$ (with $O_1 \cap O_2 = \emptyset$) are physically constrained such that $I_1$ and $O_1$ are separated from $I_2$ and $O_2$ with very precious communication resources between them. We are asked to fulfill $C$ by a minimum number of communication channels bridging such a separation.

**Planned Approach**

We tackle this problem according to the following steps.

1. Combinational instances with single output.
2. Combinational instances with two separated outputs.
3. Combinational instances with two sets of separated outputs.
4. Sequential generalizations (more specifically, for finite state machines).

![Figure 1](image-url)

**Current Status**

For Step 1, if we restrict the communication to be unidirectional as in Figure 1(a), then the minimum communication bits can be derived using functional decomposition. For the bi-directional case in Figure 1(b), minimum communication can be achieved by generalizing non-disjoint functional decomposition. For Step 2, Figure 1(c) shows the most general topology, where combinational cycles may occur. We showed that combinational cycles are necessary in certain instances to achieve minimum communication objective. We conjecture the additional power of combinational cycles stems from the fact that it must have unobservable oscillation and/or bi-stable behaviors inside circuits. Currently, we are proving the computed result of an exact algorithm is indeed the best one can achieve (assume two parties $A$ and $B$ can only have combinational computation power). Later on we may propose heuristics for practicality.
concerns. Also, we are proving the mentioned conjecture. In the next step, we will consider sequential cases.

**Processor Modeling with YAPI**

*Sam Williams*

Philips uses process networks to model their multimedia applications through a toolkit. YAPI, a UCB project, is an extension to these models, and is implemented as a Metropolis library. There are many advantages in using Kahn process networks to model systems including determinism, flexibility, concurrent processing, concurrent programming and explicit communication. All of which can be useful when modeling a processor. Furthermore, the object-oriented style allows for creation of extensible and configurable classes to represent anything from instruction sets to pipeline stages.

The goals of this project are to accurately, effectively, and easily model processor microarchitectures using the YAPI library. Of course, to model a processor, YAPI is not necessary. However, to accurately model timing, YAPI channels, which are basically unbounded FIFO's, are extremely useful, and flexible. Basic unpipelined processors are trivial so it was skipped. More advanced microarchitectures using process networks will be designed to implement pipelining, hazard detection, branch prediction and resolution, out of order execution, and decoupled microarchitecture. Furthermore, the flexibility afforded by the Java like language semantics allows for parameterized microarchitectures. For example, an execution pipeline depth could be modeled via a YAPI unbounded queue with two additions: flow control to ensure it maintains a fixed length, and initial filling with garbage up to the pipeline depth.

In order to accomplish these goals an series of microarchitectures will be constructed using YAPI, metropolis, and SystemC. All of these are required to run programs since the compilation flow starts with the YAPI extensions to metropolis, compiles it into java, converts it into systemc/c++, and final into machine language. The first step was to specify an instruction set architecture to be used throughout the examples. With this in hand, example microarchitectures can be conceived, implemented, simulated, and analyzed. Implementation can vary from a single process, which performs all computation, and a feedback channel designed to mimic the execution pipeline and thus latencies, to a multi-process/fixed length multi-channel network for out of order execution, to a fully asynchronous decoupled microarchitecture with unbounded FIFOs. Furthermore, the parameterization of all classes allows for exploration of the microarchitectural space with the ability to quantify the performance of various microarchitectures. Thus a parameter like execution pipeline depth could be used to add latency to instructions, but could also allow a cycle time to be estimated from the depth. Additionally, an area function could be derived to provide additional feedback in analyzing the design.

So far, not only has the instruction set been defined, but 5 examples have also been created. The instruction set contains only RISC like operations such as load, store, add,
subtract, boolean, etc... In addition two branches (if zero, if not zero) are defined. The architecture defines a single 256x32b register file, and always includes a Harvard style L1 cache.

The first example is a 5+n stage (modeled with 4 processes) in order statically scheduled processor. It is a very conventional looking DLX design with the additions of configurable execution pipeline depth, and configurable memory pipeline depth. As the first example, the most notable deficiency in process networks became visible - the possibility of deadlock. Without very careful planning of the architecture and scheduling within an iteration of read and write operations, it is quite easy to deadlock the machine. The second example was an abstraction of the first with the addition of a semantic for read/write operations - at the beginning of each iteration, write data from the last iteration is written (thus avoiding initial deadlock), then all ports are read. After that, the computation is performed. Thus each iteration is in the form: write, read, execute. Additionally, this example was abstracted into a single process with a feedback channel. The length of the FIFO represents the depth of the processor. The third example takes the previous single process DLX and adds hazard detection. The previous example was statically scheduled by the compiler, which highly simplifies the hardware at the expense of predictable execution models. The fourth version adds branch prediction. In this case, the processor is still an inorder design of depth n. A function (which could have been extended to a BTB) predicts branch targets and allows execution to continue. The branch is sent down the pipeline, and when it is time to commit, the branch direction is resolved and compared to the predicted direction. If a mispredict occurred, then the next n instructions which commit must be nullified, and the program counter is reset based on the correct branch direction. Any number of predicted branches may be in flight.

The most recent example is that of an out of order single-issue machine. In this case the design was partitioned into the following processes: Fetch, ReadWriteIssue, MUStation, and a parameterized number of integer execution units IUStation. Each station is a reservation station with a parameterized number of stations, and execution pipeline depth. The design is based on virtual register allocation and renaming. Branches stall, since inorder commit has not been implemented. The design, although it has an arbitrary number of execution units each of which must write through a channel to all other execution units, and the commit stage, is not particularly complex since ports may be arrays. Thus a for-loop is used to access all ports. Additionally, this design uses a ReservationStationEntry derived from class Object to pass down all channels, greatly reducing the complexity of the design. Memory aliasing is avoided via compiler support with memory sync instructions designed to avoid hazards. Additionally, memory latency can be a stochastic process by simply inserting bubbles into the memory execution pipeline to mimic the unknown latencies required to access various levels of the memory hierarchy in the presence of TLB, L1, L2, L3, page misses.

Future examples will be extended to fully utilize the process of Kahn process networks, and parameterization, and possibly self-realization of architectures.
Communication Based Analog Design
Yanmei Li

Background:
Communication Based Design (CBD) has been actively studied in the last few years as a basic design paradigm to provide design reusability and system integration at the highest possible way. Its basic principles can be applied to analog design as well, i.e. CBD-A. At the analog level, communication deals with electrical properties of single port connections rather than with complex behaviors that represent protocols between communicating blocks. As is known, a crucial point in CDB is the synthesis of communication adapters. Incompatible protocols between two communicating blocks are captured using some formal specification mechanism and an adapter is possibly synthesized and inserted without further modification of the incompatible blocks. Such adapters enable the interaction between different blocks which have behavior mismatch. In CBD-A, a similar approach is considered. A formal description of the analog interfaces has to be determined. Considering an analog port as a two-wire connection, a set of electrical properties has to be determined to capture the issues involved in the adaptation process. As an example, port features may include DC level, dynamic range, driving/loading capabilities, biasing requirements, temperature dependency, and so on. However, it is evident that these electrical features of ports alone are not sufficient to determine the adapter to be inserted between analog circuits. Since each analog circuit acting as an adapter will insert some degree of degradation on the signal (e.g. SNR degradation), some knowledge on the system itself is required to determine the specifications on the adaptation block in terms of inserted noise, distortion, CMRR, PSRR and so on.

Goal:
This project aims to explore CBD-A methodology based on an available hardware platform, Anadigm Field Programmable Analog Arrays (FPAA). For the analog adapters, we choose to focus on a particular class that performs linear transformations of the electrical signal, such as amplification, filtering and translation (DC offset). This class of adapter is indeed very useful, covering most of the signal conditioning circuits that are used in sensor read-out paths and a broad range of linear circuits. From the implementation point of view, we further restrict our implementation to OpAmp-based circuits and, more specifically, to switched capacitor circuits. Actually, very many of analog circuits can be implemented with switched capacitor structure, which can achieve high performance (e.g. it is easy to realize stable and accurate time constant) in small area. Thus, we will use Anadigm FPAA whose pre-built blocks are switched capacitor circuits to implement analog interface adapters. To achieve that, some simulation strategy based on FPAA models will also be developed to ensure fast and accurate estimations of the non-ideal effects introduced by specific FPAA configuration.

Investigative approach:
First, since we do not know enough about the FPAA performance and its signal degradation in terms of noise, distortion, dynamic range, etc, we need to get necessary data through direct measurements on suitable example circuits. Having these important data, we will develop behavioral models for the FPAA platform with some tools like Matlab, Simulink, Verilog-A. Then, with the idea similar to the formalism in CBD, we will formalize the analog interface specification from the electrical point of view and also formalize signal degradation. After that, to implement the analog adapters with FPAA, a synthesis mechanism will be developed to map analog system description and interface specification onto FPAA platform. Finally, this proposed CBD-A methodology will be tested by applying it to some simple analog design.

**Progress:**

- Literature search. But it seems that few related work was done. (We still keep searching useful papers.)
- Got familiar with the architecture and design environment of Anadigm FPAA. Its unique feature is to implement analog functions in reconfigurable architecture. So, it allows us to integrate analog circuits into programmable systems, thus gives us the analog equivalent of FPGAs. Consequently, time to solution can be drastically reduced and design flexibility is improved compared with an analog ASIC or a discrete implementation. Another important feature is, without interrupting the operation of a system, FPAAAs can be under real-time control of the system. That is because an analog design implemented with FPAAAs can be translated into C-code such that it can be controlled by a microprocessor. The basic architecture of an FPAA consists of pre-built Configurable Analog Modules (CAMs), which provide common analog elements, such as filter stages, amplifier stages, summing/difference stages, voltage multiplication, rectifiers, oscillators, references, etc. The design of complex analog systems is simplified since the design process is moved from the component level to the functional level. With FPAAAs, designers can describe analog functions like gain stages and filters without considering the lower level of components such as Op-Amps, capacitors, resistors, current mirrors, etc. Physically, this Anadigm FPAA is based on a CMOS-based fully differential switched-capacitor technology with an analog switch fabric. Here, RC-equivalent networks are provided via switching capacitance. And the FPAA can achieve a very broad range of RC-based functions. Its unique feature, analog programmability, is also provided by this switched capacitor technique.
- Studied the analysis methods for switched capacitor circuits and associated non-ideal effects (e.g. noise generation).
- To estimate non-ideal performance of FPAA (such as noise, distortion, etc), I built several example circuits and performed measurements on them. The measurements are downloaded directly from the digital oscilloscope such that the data can be processed and analyzed with the help of Matlab or in Labview environment. So far, some measurement data have been obtained, and they are being processed with Matlab.
Pupil Detection and Tracking System
Lior Zimet and Sean Kaos

Background

Eye tracking using computer vision techniques has the potential to become an important component in future human-computer interfaces. Information about the eyes can be used to detect and track human faces and bodies, which has many applications in face recognition, monitoring human activity, stereoscopic displays, and more. The purpose of the system is to find the two-dimensional location of a person’s eyes using multiple light sources and an image sensor. The final implementation will be an embedded system, and will focus on aspects of robustness, real-time performance, and cost.

When a light source is aligned with a camera’s optical axis, the camera is able to detect the light reflected from the interior of the eye and the pupil appears bright in the image. This is often seen as red-eye in flash photography. The system consists of an image sensor with a good response in the near IR region, and two IR sources with single wavelength. The light sources are placed symmetrically around the camera’s optical axis in two concentric rings. The inner ring is close to the camera optical axis and generates the bright pupil image. The outer ring diameter is sufficiently large in order to generate a dark pupil image. Figure 1 depicts the sensor and light sources configuration. The light sources are synchronized with the sensor frame sampling rate such that odd frames are illuminated by the inner ring and produce bright pupils, while even frames are illuminated by the outer ring and produce dark pupils. We can find the location of the pupils by finding the difference of odd and even frames, followed by a threshold and tracking techniques.

![Figure 1 - System Functional block diagram](image-url)
Project Goal
The goal of this project is to exercise the entire process of embedded system design from functional specification to implementation alternatives. We will describe the system by using a formal specification language, apply an appropriate model of computation, and design the architecture. We will explore different implementations considering software/hardware partitioning. Furthermore, the project will produce a viable embedded device or model of one that is capable of finding pupil locations. The design will focus on aspects of robustness, real-time performance and cost.

Approach
We plan to design the system according to the design methodology outlined in the course. Formal specifications will utilize Unified Modeling Language to describe the functional characteristics of the system with a Use Case Diagram, Sequence Diagrams, and a Class Diagram. We will then use Simulink to create a synchronous dataflow model of computation of the system. The model will be fully functional and we will simulate the system using previously captured images from various environments. This verification process can eliminate specification and functional errors before any implementation. There are no available tools that are capable of automatically synthesizing HDL from Simulink or Matlab models with video frame data structures. Therefore we will explore system architecture options manually and implement one of these options. Given enough time, we would like to compare performance of different hardware/software partitions. The system will be implemented as HDL for hardware portions and standard programming languages for software portions. Available hardware programmable platforms will execute hardware components and a possible combination of microcontrollers or a PC may execute software components.

Work progress
The following work has been done so far:

- Writing a detailed system specification in English as a base for the formal specification language description
- Describing the system in UML high level description language:
  - UML use-case diagram for the system as depicted in figure 2
  - UML Sequence diagrams
  - UML class diagram as depicted in figure 3
- Preparing Simulink model of computation for the complete system.
- Capturing frames from a system in IBM Almaden center to be used as inputs to our simulation model in Simulink and Matlab
- Writing and simulating the detection algorithm in Matlab for design and verification purposes
- Identifying major hardware components that will be used for the implementation of the system according to the system specification and constraints:
  - Sensor will be a Zoran CMOS sensor
  - Programmable FPGA board for the main dataflow and control logic
Use Case: 1. User activates the system.
2. User may modify system characteristics.
3. System synchronizes illumination with image capturing.
4. System calculates pupil position based on captured images.
5. System outputs position based on output format.

*Invariant*
(On-axis and off-axis LEDs may not be lit at the same time.)

Figure 2 - UML Use-Case diagram

Figure 3 - UML Class diagram