1. ABSTRACT
The Codesign Finite State Machine [1] (CFSM) formal model provides a suitable approach for the description of hardware/software systems. The POLIS tool from Berkeley implements the CFSM methodology but currently relies on the textually based Esterel specification language as a high level for the description of individual CFSMs. The designer must then use the Ptolemy simulator to interconnect the CFSM network and perform co-simulation. This paper describes work in progress in developing a system which instead aims to use StatemateTM, a statechart based tool for seamless specification and co-simulation of the entire CFSM network, whilst using the POLIS tool for 'C', VHDL code generation and performance estimation. This technique should give the clear advantages of using a graphical specification language together with a uniform co-simulation framework.

1.1 Keywords
Statecharts, POLIS, CFSMs

2. INTRODUCTION
Statecharts[2] are a proven and industrially desirable specification methodology. They should also be suitable for use as a specification language for a hardware/software codesign system. CFSMs use a locally synchronous, globally asynchronous paradigm that has advantages for partitioning a system. Our work has investigated how individual Statechart hierarchies can provide a synchronous model which may represent individual CFSMs in a design. Statechart represented CFSMs may then be connected in a suitable network to form a suitable system model. The Statemate tool may then be used to represent the globally asynchronous paradigm allowing the CFSM network to be simulated.

Several examples of the use of statecharts for automatic hardware and software generation exist. The work of Druinsky and Harel[3] developed techniques for mapping Statechart trees onto programmable devices, and has been developed and used by Buchenrieder[4] since. Several commercially available systems (Statemate and SpecChart code generators) exist, but from a hardware generation aspect require the use of behavioural VHDL as an intermediate step. In his paper[4] Buchenrieder states: ‘The StatemateTM code generator produces behavioural VHDL which is often not synthesisable’. The author has verified this whilst attempting to synthesise seemingly trivial Statechart models to hardware via Statemate generated VHDL, using the Cadence DFWII™ software suite.

From a software code generation aspect, to the authors knowledge, none of the existing Statechart based methods and tools allow for code performance estimation on a target processor. This is a major drawback if the tools are to be used as the basis for a codesign system. Similarly from a hardware performance estimation point of view (with the exception of Buchenrieders approach) hardware performance can only be estimated via VHDL.

The Codesign Finite State Machine approach to hardware/software codesign has several benefits. The model is globally asynchronous and this greatly simplifies partitioning of the model into hardware and software modules. Furthermore due to the relatively low level structure of individual CFSMs synthesis of hardware or software is relatively straightforward. The CFSM software synthesis uses the S-Graph as an intermediate step to estimate the performance on a target processor. This process has the advantage of producing portable and efficient code.

However since the CFSM model is at too low a level to be used directly by designers, higher level specification languages are necessary. Suitable languages are the synchronous class of languages which include Esterel[5], statecharts or a subset of VHDL.

The POLIS [6] system developed at the University of California, Berkeley, implements a HW/SW codesign system using the CFSM model as its basis. As the input to the system the intermediate level SHIFT language is used. SHIFT is capable of describing a hierarchical network of Codesign Finite State Machines. It allows for the description of individual CFSMs as reactive finite state machines. Individual CFSMs may then be embedded in a net-list which may reference other CFSMs or arithmetic, boolean or user-defined functions.

The chosen specification language for the POLIS project is Esterel[5]. To complement POLIS an strl2shift converter
program has been written, and in this way CFSMs may be developed in Esterel, converted into SHIFT and input to the POLIS program for synthesis. Whilst Esterel is a sound synchronous language with a good debugger, it does not offer the design advantages of a good graphical language such as statecharts. There is also a further drawback as having first entered an Esterel description of each CFSM the designer must next use the separate Ptolemy tool to interconnect CFSMs and perform co-simulation.

3. USING STATECHARTS TO MODEL COMMUNICATING CFSM NETWORKS

Using our methodology each individual CFSM in a network of interconnected CFSMs is modelled by a separate statechart hierarchy. The interconnections between statecharts (in the form of propagated events and signals) may be specified using the Statemate statechart package’s Activity Charts as illustrated in Figure 1. Flows between statecharts are used to represent events and signals. In this example of a paint drying system the Statecharts PAINT_SYS_CTRL and BLOWER_SYS are referenced as providing the behaviour for each CFSM.

3.1 Modelling Asynchronous CFSMs in Statemate

Statemate uses Activity Charts to allow the modelling of complex systems composed of many activities, each of which contain a behavioural specification in the form of a Statechart. The interaction of multiple Statechart hierarchies is controllable via start, stop, suspend and resume commands which may be issued by controlling Statecharts. The CFSM network must eventually be implemented in a combination of software and hardware. In the case of software a scheduling policy must be implemented on the target system. We have therefore modelled a basic scheduler in Statemate, allowing control of the interaction of events between various CFSMs.

The scheduler consists of a Statechart (Figure 2) which implements the functionality of a scheduler and controls the buffering of system inputs and outputs. Every simulated clock cycle, the scheduler triggers a Statemate Activity which detects any system input changes and places them in an input buffer modelled in Statesmate language using an array. This activity is continuously executed THROUGHOUT the system being in state IDLE. After each execution of this activity the Statechart monitors the input arrays for changes in input signal corresponding to each CFSM. If any changes are detected then the scheduler algorithm is run. This scheduler algorithm is specified by the statechart DO_SCHEDULE.

The simple algorithm presented here in Figure 3 allows for scheduling of just two activities but could be expanded for more or possibly 'n' activities.

A static reaction (code triggered when entering a state) within state SET_CURRENT_AC chooses the next activity to be scheduled in a 'round robin' fashion. Two transitions then use the Statemate rs!(A) 'resume activity' command to execute the given task if it currently has pending input event(s).
The GEN_IPS state triggers a Statemate activity which propagates pending input events to the relevant activity being scheduled. The system remains in this state until all events/signals have been propagated. The system then enters the DO_ACTIVITYn state wherein ACTIVITYn is executed for a simulation clock cycle. The Statemate toolset supports two 'time models' and in this case if its Asynchronous time model option is selected, one clock cycle corresponds to as many simulation 'micro-steps' so as to result in a new stable system stable state. So resuming execution for one cycle results in a single transition in the selected activities statechart. Whilst the chosen activity executes another activity DETECT_OPS is enabled to detect output changes in the selected statechart and store them in an output signal buffer. Once the chosen activity has executed for a simulation cycle the activity is suspended. The activity PROP_SIGS is next executed which effectively propagates any pending active outputs in the output buffer back to the relevant input buffer (so that signals may be exchanged between the scheduled tasks). If any active pending signals are then found in the input buffer the DO_SCHEDULE state is re-entered and the scheduling algorithm is repeated, otherwise the state is exited and the system enters the IDLE state detecting any further system inputs.

3.2 Embedding performance data in the statechart model
The above method allows statecharts to be used to simulate a CFSM network. At present by using tools we have developed it is possible to generate a CFSM description in the Berkeley SHIFT format from the Statechart description of the CFSM network, and use the POLIS system to perform performance estimation and synthesize software and hardware. In future work we aim to embed performance data back into the Statechart model for use in co-simulation.

4. GENERATION OF A SHIFT SPECIFICATION FROM A STATECHART MODEL
In order to use our methodology of representing a CFSM model using Statecharts in a codesign system, we interfaced it to the POLIS package to facilitate performance estimation and hardware/software synthesis. Since the POLIS system supports Esterel we initially considered generating Esterel code from our statechart model.

4.1 Generating Esterel from a statechart model
Representing a statechart design using Esterel is not as straightforward as might be thought. This is mainly due to the fact that a statechart transition may be triggered by either event occurrence, a condition variables value or a combination of these two. This in effect means that any state in a Statechart having transitions labelled with more than just basic events, must be represented in an Esterel program using a loop which, continuously checks the status of the condition variable. Furthermore in order to devise a methodology suitable for automated code generation it was necessary to explicitly encode statechart state variables in the Esterel code. This is because statecharts allow transitions directly into sub-states of a state and visa versa, and this is difficult to represent naturally in an equivalent Esterel program.

Taking these considerations we devised a suitable methodology for the generation of Esterel from statecharts. Unfortunately when this was applied to simple examples, although the Esterel code could accurately represent a statechart model, the resulting SHIFT description (obtained using the POLIS strl2shift utility) was unreasonably large. This results in poor performance when hardware or software is synthesized. For this reason using Esterel code as an intermediate step has been discounted.

4.2 Direct generation of SHIFT description from a statechart model
The statechart language provides a large number of different functions and operators that may be used for transition expressions. It is our aim to initially provide support for a reasonably large subset of these in our statechart to SHIFT converter. The system we have developed initially does not handle AND states although we have some ideas of how these could be added in the future. This is not as much a drawback as might be thought however since similar concurrent behaviour can be obtained using multiple Statecharts in an asynchronous manner.

Our converter can handle most common statechart EVENT/CONDITION expressions, including comparisons involving data items.

4.2.1 Handling Hierarchy
When generating the SHIFT representation we flatten each statechart into an FSM representable in the SHIFT language.

Hierarchical transitions are probably the major advantage of the statechart language. Our system considers any transition from a non-basic state as implicitly representing a transition from each of the source state’s child states. The target state for all transitions must also be represented in the FSM as a basic state so if the transition arrives at a non-basic state that state’s default transition is followed. The process of recursive descent is followed until a basic state is found.

4.2.2 Handling transition expressions
Statecharts generally include relatively complicated transition expressions. The use of pure valued or conditional expressions without any event information is also quite common. The CFSM methodology on the other hand assumes that all pure valued signals carry an associated event presence signal. This normally gives an advantage as the CFSM can wait for an event related to a change of valued signal or data item before reacting.

There are however instances where the CFSM must ‘self trigger’ to be able to correctly handle pure conditional expressions. The Statechart in Figure 4 illustrates such an
In this example when implementing this Statechart in a CFSM when the CFSM transitions from HEATER to PURGE it must emit a self trigger event to subsequently re-trigger itself. This is necessary due to a need to check if the HEAT value is already false, which would mean that the CFSM must transition immediately back to the HEATER state.

Two commonly used functions in Statecharts are the tr(C) and fs(C) which sense when a condition became true or false. In order to provide such functionality in the CFSM it is necessary to provide double buffering of condition variables which are used in these functions. The CFSM thus generates the ‘previous value’ of each such input on each transition. These outputs may then be fed back to the CFSM and used at the next time-step. Additional transitions must also be provided so that if the CFSM is idling in a given state then these inputs are still double buffered (even if there is no resulting transition to another state).

To greatly simplify the evaluation of Statechart expression we have used the following rules when generating the CFSM:

- Atomic transitions (those whose expressions are a single event or condition) are handled by feeding the relevant input directly to the relevant CFSM input.
- All remaining transitions are assigned their own separate pure boolean CFSM input and the transition expression is evaluated using a series of combinational functions in the CFSM net, and fed to this input.

4.2.3 Resolving transition priority
The CFSM must correctly resolve the priority of statechart transitions. This is simply achieved because each transition has one and only one input that is used as a trigger. Hence it is easy to resolve transition priorities such that low level transitions can only occur when higher level transition trigger functions are not currently active.

In the case where two transitions existing with the same priority are non-deterministic our converter currently ensures deterministic behaviour in the resulting CFSM by making the two transitions mutually exclusive.

5. USING THE SYSTEM
To date we have completed the described Statechart-CFSM converter and tested it with simple examples. Synthesis of software is satisfactory and we are able to run the POLIS generated code on a UNIX™ workstation.

The size of the resulting SW/HW (from the POLIS software) via our system compares very favourably with that from Esterel code and the appropriate converter. This is proven by a comparison (see Table 1) where we have hand-coded an equivalent description of a statechart in Esterel. Generally our system achieves more efficient results than the Esterel specification when specifications have many non event expressions within them.

<table>
<thead>
<tr>
<th>Specification method</th>
<th>POLIS SW synth. time (SPARC 20)</th>
<th>SW Costs for 68hc11 (see key)</th>
<th>HW Costs(^a) (see key)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Esterel</td>
<td>800 sec</td>
<td>size: 3911</td>
<td>POLIS failed to synth. within 4hrs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>min t: 338</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>max t: 1128</td>
<td></td>
</tr>
<tr>
<td>Statechart</td>
<td>8 sec.</td>
<td>size: 843</td>
<td>pi: 41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>min t: 223</td>
<td>po: 6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>max t: 511</td>
<td>lat: 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>sop: 353</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fac: 235</td>
</tr>
</tbody>
</table>

Table 1: Comparison of HW/SW synthesis

Key for synthesis costs
- \( size = s/w\) size in bytes
- \( min t =\) min cycle time in seconds
- \( max t =\) max cycle time in seconds
- \( pi/po =\) no. primary inputs/outputs
- \( lat =\) number of latches
- \( sop/fac = no. literals in sum-of-product or factored form\)

Synthesis of hardware using the POLIS system currently raises some problems. Due to the relatively high ARITHMETIC complexity of such CFSMs we are experiencing unreasonably long hardware synthesis times. The reason for this appears to be that the POLIS system is attempting to flatten all parts of the resulting network and then apply boolean minimisation.

One possible solution to this problem may be to use POLIS generated behavioural VHDL together with commercial synthesis tools. The improved generation of VHDL from the CFSM network is a feature of the new release of the POLIS software.

6. CONCLUSIONS
Statecharts are a convenient specification methodology for
use in conjunction with CFSM theory for the description of individual CFSMs. As they are graphical they are easier for the designer to visualise, and using tools such as Statemate offer powerful simulation capabilities.

Our methodology promises several advantages over the standard use of POLIS with the Esterel language for specification and Ptolemy for co-simulation as a codesign system:

- For the statechart models that we have considered, the resulting the CFSMs are smaller than those obtained standard via Esterel with the POLIS system. In practice this means greater efficiency in S/W and H/W implementations. This is demonstrated by the metrics in Table 1.
- Statecharts are in our view a more industrially desirable specification methodology than Esterel, being graphical in nature.
- The SHIFT file generated from a statechart is close to the original statechart specification. Therefore it may be possible for POLIS generated performance data to be embedded in the original Statemate model in order to provide high-level co-simulation in a uniform environment.
- If the specification and simulation tasks can be integrated a uniform codesign system will be obtained.

It is quite early in our project and at present we only give an outline of the system we aim to produce. To date software to generate the SHIFT code from a Statechart model has been developed. Future work will further develop the use of the Statemate environment for high level co-simulation including performance estimation. We also intend to expand our Statemate based scheduler to allow the simulation of hardware as well as software CFSMs.

Our codesign system will then be proved with a suitable real-world industrial case study.

7. ACKNOWLEDGEMENTS

The authors’ would like to thank support given from the EPSRC Engineering Design Centre, University of Newcastle upon Tyne during undertaking this work. Also thanks to Luciano Lavagno, Bassam Tabbara and the POLIS support team for their advice.

8. REFERENCES