Extending POLIS with User-Defined Data Types
EE 249 Final Project

by

Arvind Thirunarayanan
Prof. Alberto Sangiovanni-Vincentelli

Mentors: Marco Sgroi, Bassam Tabbara
Introduction

POLIS is, by definition, a co-design environment for system design of control dominated embedded systems. It’s used for exploration of system architectures and function architecture co-design. The co-design environment of POLIS is used to create mixed systems, such as those involving both hardware and software components. The design process itself includes various steps of design including initial user specification of the system, final synthesis of the system design, as well as estimation and verification of the design.

The goal of this project was to modify the POLIS tool, extending it with support for user-defined data types. Since the POLIS tool functions at several levels of abstraction, beginning with user interaction at the top levels, and synthesizable output at the bottom, this support had to be incorporated through each stage of the design.

Design Flow in POLIS

When designing a system in POLIS, there are two paths to follow: the simulation path and the synthesis path. Simulation allows a design to be tested and explored thoroughly before committing it to the final synthesis path. This allows the user to test various configurations and decide on an optimum design to meet the system objectives.

The user develops the initial specification for the embedded system in a high level language. POLIS currently uses Esterel to write each module of the specification. The modules are then compiled into POLIS SHIFT files. SHIFT is the internal POLIS Software Hardware Intermediate Format used to specify the model of computation, a set of interacting extended
finite state machines, or CFSMs, representing each module, and the CFSM associated with it as files.

Simulation Path

Once the initial specification is completed and the design compiled into the SHIFT files, the design can then be read into POLIS. The internal S-graph representation of the design is then created and optimized.

At this point, the user chooses a target microprocessor to eventually run the software in the final implementation. The user can also run estimation tools, using pre-existing parameter files that describe processor characteristics, to get an idea of the code size and execution speed of the resulting software.

Finally, POLIS generates the C-code and Ptolemy code for each module in the design. The generated module code can then be read into a simulator like Ptolemy outside of POLIS to do a functional and performance simulation of the system. This simulation includes validation to analyze a design’s performance and efficiency. Within the simulation, the user can also examine the system interconnections and explore the effects of changing design parameters. The user can also re-simulate the system after trying different ways to re-partition the design, choosing a hardware or software implementation for various parts of the system to determine and evaluate performance.

Synthesis Path

Once the user is satisfied with the simulation, the design can then be read back into POLIS. At this stage, the partitioning decided upon in the simulation stage is actually implemented.
For the parts of the design that will be implemented in hardware, POLIS is used to translate and optimize the internal format into a synthesizable hardware format like VHDL or BLIF (Berkeley Logic Interchange Format). These are the netlist files that actually describe the hardware implementation of the FSMs.

For the parts of the design that will be implemented in software, POLIS is again used to create and optimize the internal S-graph representation of the design. Similar to the simulation phase, POLIS generates the C-code for each module. POLIS also generates the real time operating system (RTOS) code that will be used on the target microprocessor to run the software portion of the design.

The final step in the design flow involves generating the actual hardware and software implementation. At this stage, the user chooses their target hardware, like a Xilinx FPGA, for example. The netlist files generated by POLIS are then mapped onto the hardware. The user also downloads the RTOS and the generated software code onto the target microprocessor or microcontroller. Finally, the two partitions are merged to produce a synthesized embedded system [1].

Project Description

One of the motivating factors behind this project is the Two Chip Intercom (TCI) project at the Berkeley Wireless Research Center (BWRC). The intercom project is aimed at creating a network of interconnected remote units and a base station all talking to each other. POLIS is one of the tools used to create the system design for the intercom project, along with Felix VCC.

The advantage of using POLIS over VCC is its synthesis path directly to the hardware level. Unlike VCC, POLIS can be used to generate the actual hardware for each module or
CFSM. But one of the major stumbling blocks to using POLIS has been the lack of internal support for any user-defined data types.

The events used for communication among modules in POLIS are currently capable of carrying data of integer types only. In the case of the intercom project, this is inherently unsuitable. Much of the necessary communication between modules involves data structures that have several fields of information. Using the current implementation of POLIS, the design would involve sending each field individually, then waiting for some form of acknowledgement from the receiving module before transmitting the next field.

There is limited support for user-defined types within POLIS [1]. However, this support is limited strictly to the software portions of the design. POLIS achieves this in one of two ways. In the case of numeric non-integer types, such as floats, POLIS allows the user to declare the new type as a special integer at the initial Esterel level. The environment then relies on the implicit conversions and casts provided by the C compiler to allow them to be used correctly. In the case of more involved data types, such as structs and unions, the declarations of the new C type and the type definitions can be entered into an external ‘.c’ and ‘.h’ file. These external files are used at simulation time and when the final executable is created for the target processor. While the second method does allow for limited data type support, it still does not allow user-defined types to be used in communication between modules. They can only be used internally within the module they are declared in. Besides, both of these are software-only solutions. Neither of them addresses the hardware synthesis path.

This is the driving factor behind this project, namely to come up with an implementation to include support for user-defined data types in the SHIFT files, and then extend this support down to the synthesis levels.
Proposed Solution

The following diagram shows a graphical overview of the design flow in POLIS.

![Diagram](image)

One solution to implementing user-defined data type support in POLIS is to allow the user to enter type definitions and declarations at the initial specification level. Given the limited support for user-defined types, POLIS uses information in an auxiliary file to determine bit allocation for the new type. The auxiliary file is also the location where a numeric non-integer type, like a float, is re-defined as a special case of an integer. The user can also define new types here that are subranges of the existing integer data type.

The idea is to let the user enter the type information in this auxiliary file in a C like format. This information would then be processed by the design environment, converting the information into a form accessible by POLIS at all the levels of the design without the user having to deal with the details of the actual implementation of the type. For example, given a
typical struct, like a complex number that has two fields, a real integer and an imaginary integer, the entry in the aux file would resemble the corresponding C construct as shown below.

```c
typedef struct {
    int re;
    int im;
} cplx;
```

![Figure 2: Type information in aux file](image)

When the initial Esterel specification files are compiled into SHIFT, this information can then be added to the SHIFT files as well. The current methods provide no support for data type information to be passed into the hardware synthesis path because they bypass the SHIFT files entirely. The external `.c` and `.h` file that currently contain the type information are used directly when generating the simulation files or the synthesized software executable.

In the case of numeric integer or non-integer types, the SHIFT files currently carry information about how many bits an instance of the data type occupies. For other user-defined types, however, there is no information in the SHIFT files as shown in Fig 3.

```plaintext
File.shift
.net net_sender
.inputs *il
.outputs *ol(ol)
.nb ol 0       ← user-defined
...
.end
```

![Figure 3: Current SHIFT implementation](image)
By adding the type information into the SHIFT files, POLIS can then extend the user-defined data type support through SHIFT down to the hardware synthesis levels. The modified SHIFT file would look similar to Fig 4.

```
File.shift
.net net_sender
.type int 16
.type struct cplx re im
.dt re int
.dt im int
.inputs *i1
.outputs *o1(o1)
.dt o1 cplx
...
.end
```

Figure 4: Modified SHIFT file

The new information added into the SHIFT file describes the new data type, and later declares an instance of the type. With this information now available in the SHIFT file, it is possible to translate the data types into corresponding ones in hardware. For instance, the struct data type might map onto the record construct in VHDL.

**Current Work**

The first task in implementing user-defined data type support in POLIS involved testing the changes implemented by Luciano Lavagno. His changes to POLIS and Ptolemy allow the use of the external type declaration and definition files to actually communicate the user-defined
types between modules. Again, this is still a software only solution. But it involved implementing a simple sender receiver network as shown below using Esterel.

![Sender/Receiver network](image)

**Figure 5: Sender/Receiver network**

The sender module awaits the signal `i1`, then emits the event `o1`, which carries data of the `cplx` user-defined type. The receiver module awaits the `o1` event, then extracts the fields and emits them as the two integers, `y1` and `y2`. The design was first simulated in Ptolemy, then compiled into a synthesized software executable. Since the implementation did not support hardware, the entire design was partitioned into software. The real time operating system simulator is shown below, running the software for the sender and receiver modules. Upon the input of the `i1` signal, the system outputs `y1` and `y2`, with the values sent to the receiver from the sender.

```
sndrcv_polis

os sim> il;
y1(2) y2(3)
os sim>
```

**Figure 6: OS simulator running Sender/Receiver software**
The next step involved the implementation details of the translation from the user edited aux file to the SHIFT files. This meant first deciding on the translation format from the Esterel files to the SHIFT files, and then editing the aux file parser functions to recognize the new user additions and the ‘writeshift’ functions to correctly translate them to SHIFT. The format of the additions to the SHIFT files was chosen to be similar to existing SHIFT syntax so that the ‘readshift’ functions that would be used later in the POLIS design flow could easily parse the generated SHIFT files.

Conclusion

The changes to the aux parsing routines have been completed, but the SHIFT writing routines are still being modified. After the changes to the aux reader and SHIFT writer functions are implemented, there are a few other steps left in making POLIS truly support user-defined data types.

First, the external type definition files that are currently hand written by the user should be automatically generated from the information in either the aux or the SHIFT files. This would restrict the user’s involvement in the system specification to only the top levels of the design, thus enabling a top down design methodology. Second, the details for the translation from the SHIFT files to the hardware netlist format need to be defined and implemented. And finally, the functions needed to access and manipulate the data types, or their fields in the case of complex types like structs and unions, need to be determined, and generated automatically as a library for use at the synthesis levels.

Once POLIS has been modified with these changes to support user-defined data types, the designs for the TCI project can then be simulated and synthesized.
References
