Towards Automatic Program Specification Using SME Models

Communicating Process Architectures 2018 – Technische Universität Dresden

Alberte Thegler,
Mads Ohm Larsen,
Kenneth Skovhede,
and Brian Vinter

Niels Bohr Institute, University of Copenhagen, Denmark

Ariane-5

4th June 1996.

Ariane-5

4th June 1996.

Total failure on launch.

Ariane-5

4th June 1996.

Total failure on launch.

Converting a 64-bit floating point number to signed 16-bit integer.

Ariane-5

4th June 1996.

Total failure on launch.

Converting a 64-bit floating point number to signed 16-bit integer.

Overflow caused the self-destruct mechanism in both primary and backup computer.

Ariane-5

4th June 1996.

Total failure on launch.

Converting a 64-bit floating point number to signed 16-bit integer.

Overflow caused the self-destruct mechanism in both primary and backup computer.

No people were harmed.

The Patriot Missile Failure

25th February 1991 in the Persian Gulf war.

The Patriot Missile Failure

25th February 1991 in the Persian Gulf war.

A Patriot missile failed to intercept an incomming ”Scud”.

The Patriot Missile Failure

25th February 1991 in the Persian Gulf war.

A Patriot missile failed to intercept an incomming ”Scud”.

Conversion of time since last boot from an integer to a real number was performed using a 24 bit register.

The Patriot Missile Failure

25th February 1991 in the Persian Gulf war.

A Patriot missile failed to intercept an incomming ”Scud”.

Conversion of time since last boot from an integer to a real number was performed using a 24 bit register.

The patriot missile missed the Scud which struck a U.S Army barracks, killing 28 soldiers.

Why should we verify hardware?

Because, as these examples have shown, the consequences of not verifying can be devastating.

Loss of milions of money.

Loss of human life.

What have we done?

A transpiler (Source-to-source compiler) which transpiles SMEIL to CSP_M in order to verify SME models with FDR4.

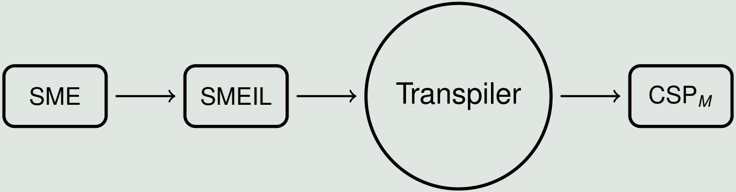

Figure. SME to CSP_M transpiler.

How do we use SME?

The SME model builds on the CSP algebra and therefore all SME models have a corresponding CSP model.

How do we use SME?

The SME model builds on the CSP algebra and therefore all SME models have a corresponding CSP model.

We transpile not only the SME network, but also all the SME processes and their content.

How do we use SME?

The SME model builds on the CSP algebra and therefore all SME models have a corresponding CSP model.

We transpile not only the SME network, but also all the SME processes and their content.

We can translate SME sequentially which simplifies the transpilation.

How do we use SMEIL?

Introduced by Truls Asheim in the previous presentation.

How do we use SMEIL?

Introduced by Truls Asheim in the previous presentation.

We transpile from SMEIL to CSP$_M$.
And then verify it in FDR4.

How do we use SMEIL?

Introduced by Truls Asheim in the previous presentation.

We transpile from SMEIL to CSP$_M$.
And then verify it in FDR4.

The transpiler currently only works with pure SMEIL programs.

Seven segment display clock

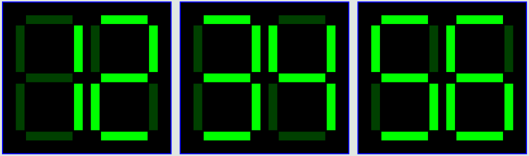

Figure. Digital clock with six seven segment displays, displaying 12:34:56.

Seven segment display clock

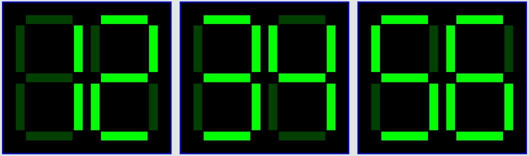

Figure. Digital clock with six seven segment displays, displaying 12:34:56.

Seconds since midnight.

Seven segment display clock

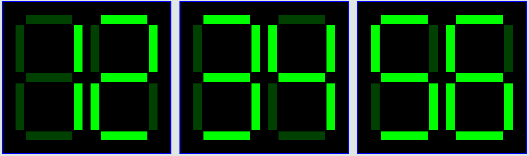

Figure. Digital clock with six seven segment displays, displaying 12:34:56.

Seconds since midnight.

Arithmetics calculate hours, minutes and seconds respectively.

Seven segment display clock

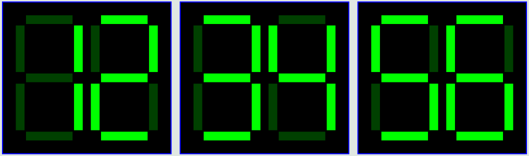

Figure. Digital clock with six seven segment displays, displaying 12:34:56.

Seconds since midnight.

Arithmetics calculate hours, minutes and seconds respectively.

Two seven segment displays pr. `time` process.

Seven segment display clock

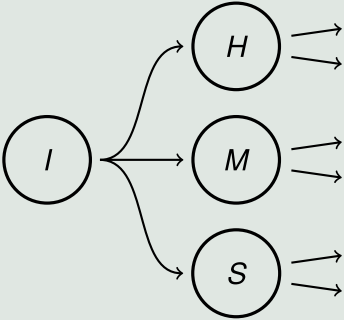

Figure. SMEIL network for a seven segment display clock. Each SMEIL process is represented by a cicle with a letter corresponding to the processes Input, Hours, Minutes and Seconds respectively.

What are we verifying?

One seven segment display can only display the numbers 0-9.
4 bits can represent 0-15, which is more than needed.

What are we verifying?

One seven segment display can only display the numbers 0-9.
4 bits can represent 0-15, which is more than needed.

We can verify that the values communicated to all the seven segment displays does not exceed the expected values.

What are we verifying?

One seven segment display can only display the numbers 0-9.
4 bits can represent 0-15, which is more than needed.

We can verify that the values communicated to all the seven segment displays does not exceed the expected values.

In this case we can restrict the assertions further.
The time will never be more than 23:59:59.

What are we verifying?

One seven segment display can only display the numbers 0-9.
4 bits can represent 0-15, which is more than needed.

We can verify that the values communicated to all the seven segment displays does not exceed the expected values.

In this case we can restrict the assertions further.
The time will never be more than 23:59:59.

In general, we verify the values comnmunicated on CSP_M channels.

Seven Segment display clock

SMEIL code:

```
proc seconds (in seconds_in)
    bus seconds_out {first_digit: u3 range 0 to 5;
                     second_digit: u4 range 0 to 9;};
    var seconds: u6 range 1 to 59;
    var seconds_first_temp: u3 range 0 to 5;
    var seconds_second_temp: u4 range 0 to 9;
{
    seconds = seconds_in.val % 60;
    seconds_first_temp = seconds / 10;
    seconds_second_temp = seconds % 10;
    seconds_out.first_digit = seconds_first_temp;
    seconds_out.second_digit = seconds_second_temp;
}
```

The transpiling

SMEIL bus to CSP$_M$ channel.

The transpiling

SMEIL bus to CSP_M channel.

CSP_M process structure.

The transpiling

SMEIL bus to CSP$_M$ channel.

CSP$_M$ process structure.

The monitor process.

SMEIL bus to CSP$_M$ channel

SMEIL code:

```
proc seconds (in seconds_in)
    bus seconds_out {first_digit: u3 range 0 to 5;
                     second_digit: u4 range 0 to 9;};
```

SMEIL bus to CSP$_M$ channel

SMEIL code:

```
proc seconds (in seconds_in)
    bus seconds_out {first_digit: u3 range 0 to 5;
                     second_digit: u4 range 0 to 9;};
```

CSP$_M$ code:

```
channel seconds_out_first_digit : {0..7}
channel seconds_out_second_digit : {0..15}
```

CSP$_M$ process structure

SMEIL code:

```
proc seconds (in seconds_in)
    ⋮
{
    seconds = seconds_in.val % 60;
    seconds_first_temp = seconds / 10;
    seconds_second_temp = seconds % 10;
    seconds_out.first_digit = seconds_first_temp;
    seconds_out.second_digit = seconds_second_temp;
}
```

CSP_M process structure

SMEIL code:

```
proc seconds (in seconds_in)
    ⋮
{
    seconds = seconds_in.val % 60;
    seconds_first_temp = seconds / 10;
    seconds_second_temp = seconds % 10;
    seconds_out.first_digit = seconds_first_temp;
    seconds_out.second_digit = seconds_second_temp;
}
```

CSP_M code:

```
Seconds(seconds_in) =
let
    seconds = seconds_in % 60
    seconds_first_temp = seconds / 10
    seconds_second_temp = seconds % 10
within
    seconds_out_first_digit ! seconds_first_temp ->
    seconds_out_second_digit ! seconds_second_temp ->
    SKIP
```

The monitor process

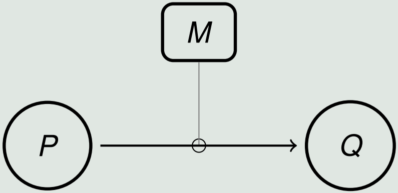

Figure. The monitor process *M* listens in on the communication between *P* and *Q* in order to assert the communicated values.

The monitor process

SMEIL code:

```
proc seconds (in seconds_in)
    bus seconds_out {first_digit: u3 range 0 to 5;
                     second_digit: u4 range 0 to 9;};
```

The monitor process

SMEIL code:

```
proc seconds (in seconds_in)
    bus seconds_out {first_digit: u3 range 0 to 5;
                     second_digit: u4 range 0 to 9;};
```

CSP$_M$ code:

```
Seconds_out_first_digit_monitor(c) =
    c ? x -> if 0 <= x and x <= 5 then SKIP else STOP
Seconds_out_second_digit_monitor(c) =
    c ? x -> if 0 <= x and x <= 9 then SKIP else STOP
```

Seven segment display clock

CSP$_M$ code:

```
channel seconds_out_first_digit : {0..7}
channel seconds_out_second_digit : {0..15}

Seconds(seconds_in) =
let
    seconds = seconds_in % 60
    seconds_first_temp = seconds / 10
    seconds_second_temp = seconds % 10
within
    seconds_out_first_digit ! seconds_first_temp ->
    seconds_out_second_digit ! seconds_second_temp ->
    SKIP

Seconds_out_first_digit_monitor(c) =
    c ? x -> if 0 <= x and x <= 5 then SKIP else STOP
Seconds_out_second_digit_monitor(c) =
    c ? x -> if 0 <= x and x <= 9 then SKIP else STOP

N_seconds = clock_out_val ? variable ->
            (Seconds(variable)
            [  seconds_out_first_digit ]
            Seconds_out_first_digit_monitor(seconds_out_first_digit))
            [  seconds_out_second_digit ]
            Seconds_out_second_digit_monitor(seconds_out_second_digit)

assert SKIP [F= N_seconds \ Events
```

Seven segment display clock

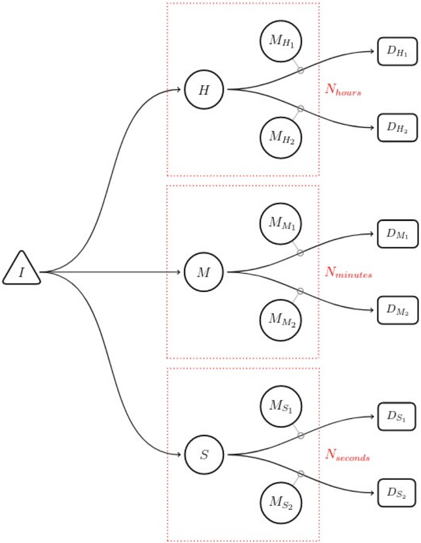

Figure. A seven segment display clock network in CSP$_M$.

Verification in FDR4

The seven segment example have been run on a Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz.

We had a few problems with FDR4, but dividing the task was possible in this case.

When dividing the task, 0 to 131071 seconds can be verified in $\approx$ 2 hours 40 minutes.

Conclusion

With this system we can transpile hardware models to CSP$_M$.

Conclusion

With this system we can transpile hardware models to CSP$_M$.

Verify values on the CSP$_M$ channels.

Conclusion

With this system we can transpile hardware models to CSP$_M$.

Verify values on the CSP$_M$ channels.

Verify the original hardware model.

Conclusion

With this system we can transpile hardware models to CSP$_M$.

Verify values on the CSP$_M$ channels.

Verify the original hardware model.

Extract specification.

Future work

Hardware/software co-simulation.

Future work

Hardware/software co-simulation.

Creating more extensive examples to show the possibilities of the system.

Questions?

Thank you!