



Faculty of Science

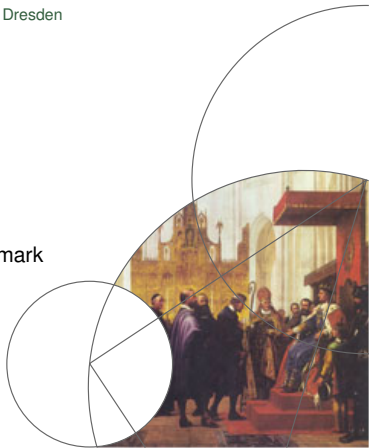


# Towards Automatic Program Specification Using SME Models

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

Picture?



# 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 where harmed



# The Patriot Missile Failure

Picture?





# 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 incoming "Scud".



# The Patriot Missile Failure

25th February 1991 in the Persian Gulf war

A Patriot missile failed to intercept an incoming "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 incoming "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 risk of not verifying can be devastating.

Loss of millions of money

Loss of human life



# What have we done?

A transpiler which transpiles SMEIL code to  $\text{CSP}_M$  in order to verify SME models with FDR4

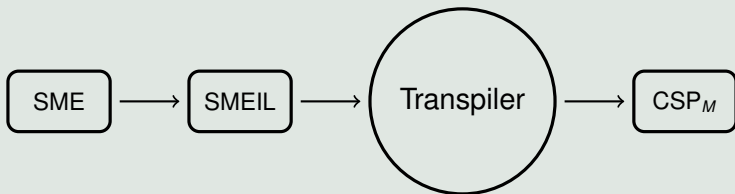


Figure. SME to  $\text{CSP}_M$  transpiler.



# 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  $\text{CSP}_M$   
And then verify it in FDR4



## How do we use SMEIL?

Introduced by Truls Asheim in the previous presentation

We transpile from SMEIL to  $\text{CSP}_M$   
And then verify it in FDR4

The transpiler currently only works with pure SMEIL programs

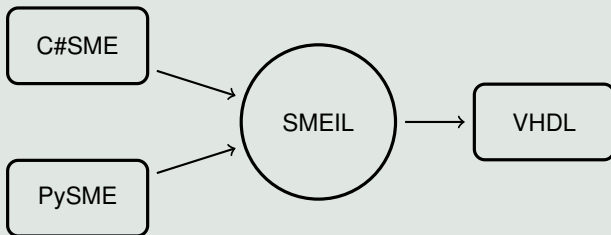


Figure. SMEIL transpiler structure.



# Seven segment display clock

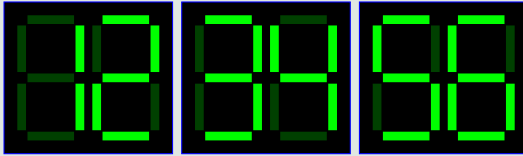


Figure. Digital clock with six seven segment displays, displaying 12:34:56.

# Seven segment display clock

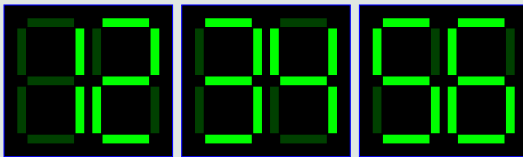


Figure. Digital clock with six seven segment displays, displaying 12:34:56.

Seconds since midnight



# Seven segment display clock

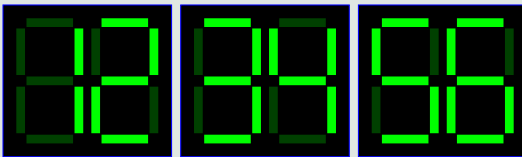


Figure. Digital clock with six seven segment displays, displaying 12:34:56.

Seconds since midnight

Arithmetics calculate hours, minutes and seconds respectively



# Seven segment display clock

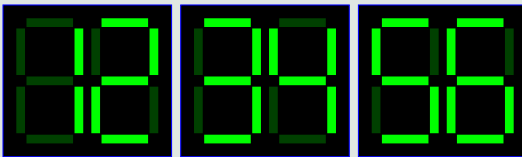


Figure. Digital clock with six seven segment displays, displaying 12:34:56.

Seconds since midnight

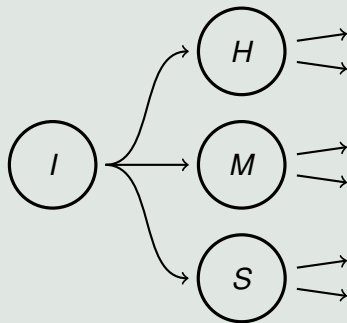
Arithmetics calculate hours, minutes and seconds respectively

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 circle 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.  
Hours will never be more than 24, etc.



# 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.  
Hours will never be more than 24, etc.

In general, we verify the values communicated on  $CSP_M$  channels



# Seven Segment display clock

SMEIL code:

```
1  proc seconds (in seconds_in)
2      bus seconds_out {first_digit: u3 range 0 to 5;
3                      second_digit: u4 range 0 to 9;};
4      var seconds: u6 range 1 to 59;
5      var seconds_first_temp: u3 range 0 to 5;
6      var seconds_second_temp: u4 range 0 to 9;
7  {
8      seconds = seconds_in.val % 60;
9      seconds_first_temp = seconds / 10;
10     seconds_second_temp = seconds % 10;
11     seconds_out.first_digit = seconds_first_temp;
12     seconds_out.second_digit = seconds_second_temp;
13 }
```



# The transpiling

SMEIL bus to  $\text{CSP}_M$  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<sub>M</sub> channel

SMEIL code:

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



# SMEIL bus to CSP<sub>M</sub> channel

## SMEIL code:

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

## CSP<sub>M</sub> code:

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



# CSP<sub>M</sub> process structure

SMEIL code:

```
1  proc seconds (in seconds_in)
2      :
3  {
4      seconds = seconds_in.val % 60;
5      seconds_first_temp = seconds / 10;
6      seconds_second_temp = seconds % 10;
7      seconds_out.first_digit = seconds_first_temp;
8      seconds_out.second_digit = seconds_second_temp;
9  }
```



# CSP<sub>M</sub> process structure

## SMEIL code:

```

1  proc seconds (in seconds_in)
2      :
3  {
4      seconds = seconds_in.val % 60;
5      seconds_first_temp = seconds / 10;
6      seconds_second_temp = seconds % 10;
7      seconds_out.first_digit = seconds_first_temp;
8      seconds_out.second_digit = seconds_second_temp;
9  }
```

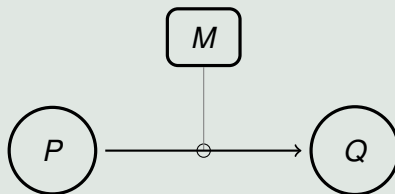
## CSP<sub>M</sub> code:

```

1  Seconds(seconds_in) =
2  let
3      seconds = seconds_in % 60
4      seconds_first_temp = seconds / 10
5      seconds_second_temp = seconds % 10
6  within
7      seconds_out.first_digit ! seconds_first_temp ->
8      seconds_out.second_digit ! seconds_second_temp ->
9  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:

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



# The monitor process

## SMEIL code:

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

## CSP<sub>M</sub> code:

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





# Seven segment display clock

## CSP<sub>M</sub> code:

```

1  channel seconds_out_first_digit : {0..7}
2  channel seconds_out_second_digit : {0..15}
3
4  Seconds(seconds_in) =
5  let
6      seconds = seconds_in % 60
7      seconds_first_temp = seconds / 10
8      seconds_second_temp = seconds % 10
9  within
10     seconds_out_first_digit ! seconds_first_temp ->
11     seconds_out_second_digit ! seconds_second_temp ->
12     SKIP
13
14     Seconds_out_first_digit_monitor(c) =
15         c ? x -> if 0 <= x and x <= 5 then SKIP else STOP
16     Seconds_out_second_digit_monitor(c) =
17         c ? x -> if 0 <= x and x <= 9 then SKIP else STOP
18
19     N_seconds = clock_out_val ? variable ->
20         (Seconds(variable)
21          [ seconds_out_first_digit ]
22          Seconds_out_first_digit_monitor(seconds_out_first_digit))
23         [ seconds_out_second_digit ]
24         Seconds_out_second_digit_monitor(seconds_out_second_digit)
25
26  assert SKIP [F= N_seconds \ Events

```



# Results - time to verify in FDR4?

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

The example were run x times and the average was measured.



# Conclusion

With this system we can transpile hardware models to  $\text{CSP}_M$ .



# Conclusion

With this system we can transpile hardware models to  $\text{CSP}_M$ .

Verify values on the  $\text{CSP}_M$  channels.



# Conclusion

With this system we can transpile hardware models to  $\text{CSP}_M$ .

Verify values on the  $\text{CSP}_M$  channels.

Verify the original hardware model.



# Conclusion

With this system we can transpile hardware models to  $\text{CSP}_M$ .

Verify values on the  $\text{CSP}_M$  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!

Feel free to ask anything.

