

Experiments in translating CSP | | B into Handel-C

Steve Schneider, Helen Treharne,
Alistair McEwan, and Wilson Ifill

University of Surrey
University of Leicester

AWE Ltd.

www.surrey.ac.uk



Structure of presentation

Context

CSP | | B

Experiments

Further developments

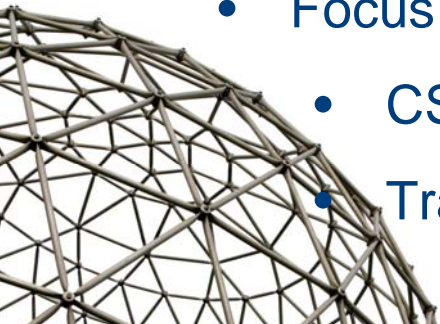


Context



Future Technologies for System Design

- Formal methods for development of hardware/software codesigns.
 - Formal modelling and analysis.
 - Implementation design and refinement.
 - Translation and implementation.
- Three year project at the University of Surrey.
- Supported, and with technical input, from AWE.
- Surrey: Steve Schneider, Helen Treharne, Alistair McEwan, David Pizarro.
- AWE: Wilson Ifill, Neil Evans.
- Focus of this project:
 - CSP||B for modelling, specification, and design.
 - Translation to Handel-C.

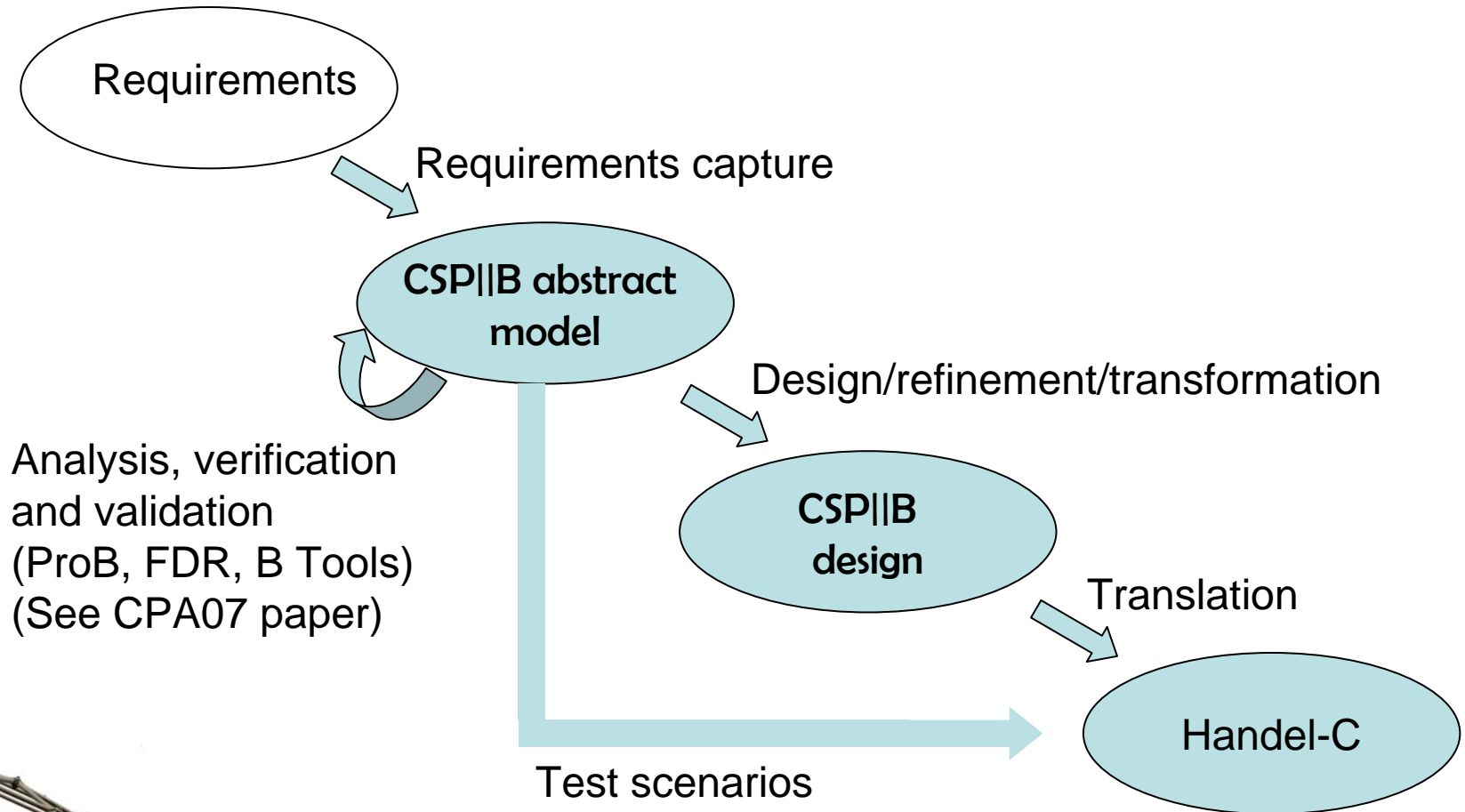


Context

- AWE have a long term interest in methods for development and verification of high assurance hardware and software systems.
- Timescale: 15 years from inception to production.
- Currently in the early years: fundamental research.
 - Investigating formal technologies.
 - Developing methodology, foundations, and techniques.
 - Our project is focusing on specific technologies to ground the research: CSP||B and Handel-C.



Development Methodology



Why CSP || B and Handel-C ?

- CSP||B
 - Provides a formal underpinning, supports modelling, specification, development, and verification.
 - Incorporates control and state within a single framework.
 - Mature industrial strength tool support.
 - CSP||B developed in Surrey: local ownership.
- Handel-C
 - Established route to hardware – a key aspect of the project.
 - Links with CSP, and appropriate target language for B.
 - Prior work on translating CSP to Handel-C.



CSP | | B



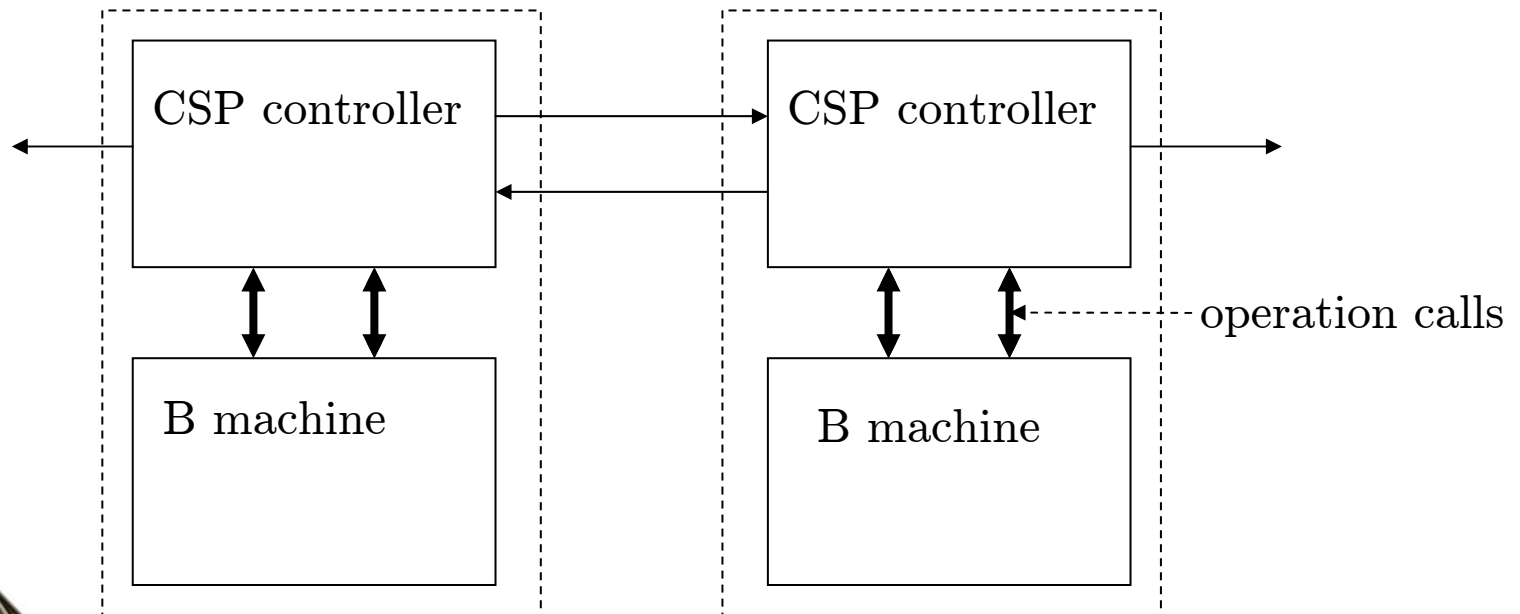
Combining CSP and B : events and state

- Separation of concerns
 - State (object based, cf Z) described in B.
 - Concurrency, communication, and control encapsulated in CSP.
- Re-use of existing tools (as well as development of new ones).
 - Retains original semantics for CSP and B.
- Rigorous semantic grounding: formal link through Morgan's failures semantics for action systems: B machines are given CSP semantics.
- Communicating abstract data types model.



CSP | B: Controlling B machines

- A controlled component consists of a CSP controller process in parallel with a B machine.
- Semantics given by CSP semantics of both components.
- A CSP event $e!v?x$ matches a B operation call $x \leftarrow e(v)$.



Example

CSP controller

```
SW_CONTROL =  
press → light →  
timeout → dark →  
SW_CONTROL
```

B machine

```
MACHINE Switch  
VARIABLES switch  
INVARIANT switch : {off, on}  
INITIALISATION switch := off  
OPERATIONS  
  light = PRE switch = off  
         THEN switch := on  
         END;  
  dark = PRE switch = on  
         THEN switch := off  
         END  
END
```



Example (with I/O)

CSP controller

SW_CONTROL =

read?n → *add!n* → *CON*

CON =

read?n → *add!n* → *CON*

□ *average?m* →

report!m → *CON*

B machine

MACHINE Totaliser

VARIABLES total, num

INITIALISATION total := 0

|| num := 0

OPERATIONS

add(nn) = PRE nn : NAT

THEN tot := tot + nn

|| num := num + 1

END;

mm <-- average =

PRE num > 0

THEN mm := tot / num

END

END

Consistency

- Operations must be called within their preconditions. This needs to be proved for controlled components.
- There are established techniques (based on wp semantics) for establishing consistency between a controller and a controlled machine.
 - Consistency expressed as divergence-freedom.
 - Divergence-freedom means operations called within their preconditions.
- Note: the previous examples are consistent.



The ProB tool: CSP || B analysis

Type checking

Animation

Walk through

Model-checking

Invariant violation

Precondition violation

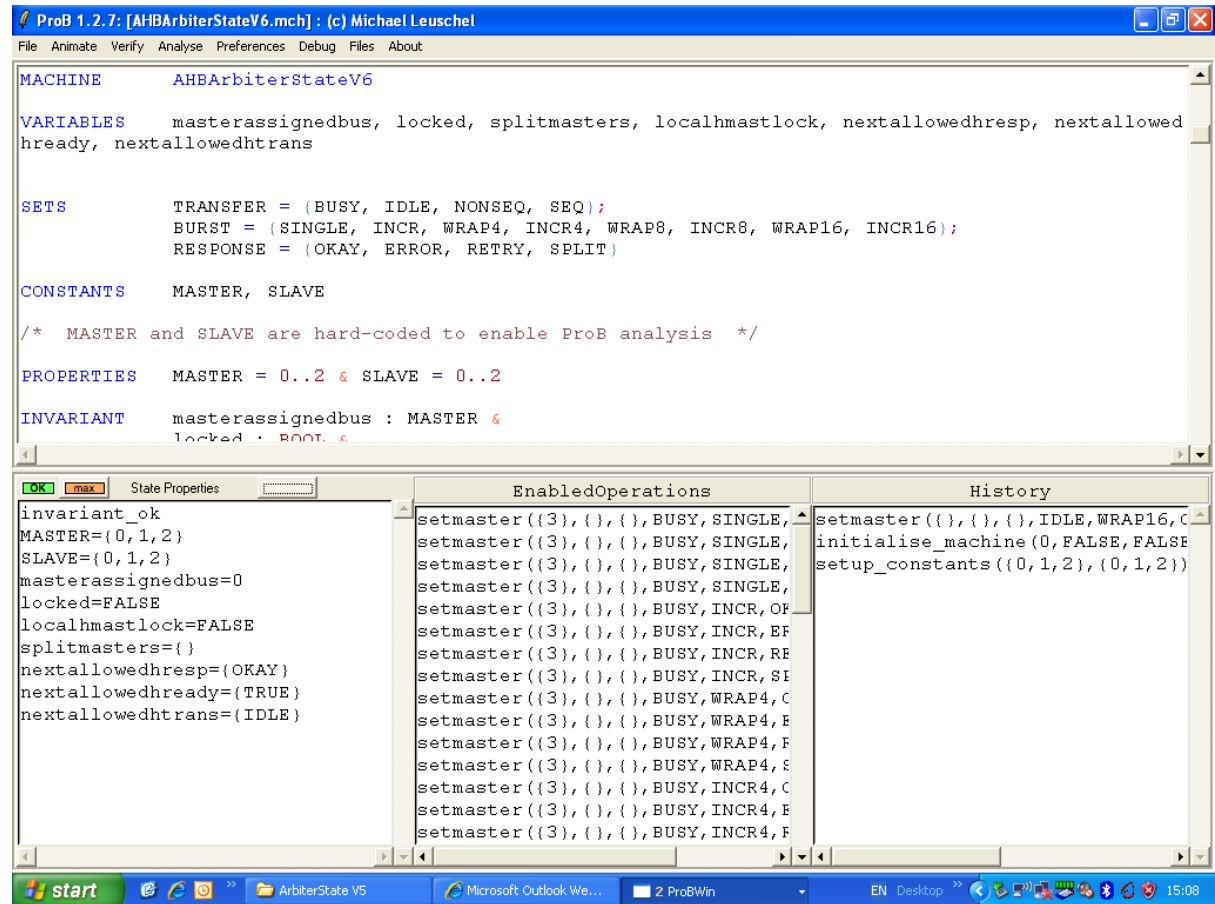
Deadlock

LTL style

Supports CSP_M

Suitable for B, CSP, and
CSP||B

[Originates from Michael
Leuschel; enhanced in
conjunction with AWE
and FutureTech]



```
ProB 1.2.7: [AHBArbiterStateV6.mch] : (c) Michael Leuschel
File Animate Verify Analyse Preferences Debug Files About

MACHINE      AHBArbiterStateV6

VARIABLES    masterassignedbus, locked, splitmasters, localhmastlock, nextallowedhresp, nextallowed
hready, nextallowedhtrans

SETS         TRANSFER = {BUSY, IDLE, NONSEQ, SEQ};
            BURST = {SINGLE, INCR, WRAP4, INCR4, WRAP8, INCR8, WRAP16, INCR16};
            RESPONSE = {OKAY, ERROR, RETRY, SPLIT}

CONSTANTS    MASTER, SLAVE

/* MASTER and SLAVE are hard-coded to enable ProB analysis */

PROPERTIES    MASTER = 0..2 & SLAVE = 0..2

INVARIANT    masterassignedbus : MASTER &
            locked : NOT &
```

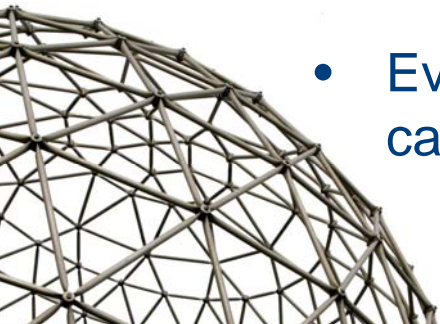
State Properties	EnabledOperations	History
invariant_ok MASTER={0,1,2} SLAVE={0,1,2} masterassignedbus=0 locked=FALSE localhmastlock=FALSE splitmasters={} nextallowedhresp={OKAY} nextallowedhready={TRUE} nextallowedhtrans={IDLE}	setmaster({3},{},{},BUSY,SINGLE, setmaster({3},{},{},BUSY,SINGLE, setmaster({3},{},{},BUSY,SINGLE, setmaster({3},{},{},BUSY,SINGLE, setmaster({3},{},{},BUSY,INCR,OF setmaster({3},{},{},BUSY,INCR,EF setmaster({3},{},{},BUSY,INCR,RE setmaster({3},{},{},BUSY,INCR,SI setmaster({3},{},{},BUSY,WRAP4,C setmaster({3},{},{},BUSY,WRAP4,E setmaster({3},{},{},BUSY,WRAP4,F setmaster({3},{},{},BUSY,WRAP4,S setmaster({3},{},{},BUSY,INCR4,C setmaster({3},{},{},BUSY,INCR4,E setmaster({3},{},{},BUSY,INCR4,F	setmaster({},{},{},IDLE,WRAP16,C initialise_machine(0,FALSE,FALSE setup_constants({0,1,2},{0,1,2})

Experiments



Translation to Handel-C

- Handel-C provides a route to hardware:
 - Contains a core subset of C (state).
 - Provides support for CSP-like concurrent behaviour (communication).
 - Clocked.
- Previous work on translating CSP to Handel-C (Stepney; Oliveira and Woodcock; Philips and Stiles; Ifill).
 - All `state' is in the CSP.
- Our aim is to implement the controlled B machines.
 - Maintain the state.
 - Events associated with B machines correspond to operation calls.



Approach

- Invent simple (artificial) CSP||B examples which contain a feature we wish to explore. Identify issues that emerge as we do the translation.
- Investigate how such examples are rendered in Handel-C.
- Example 1: a first CSP||B component: translating the CSP and the B together.
- Example 2: parallelism in the CSP controller.
- Example 3: data refinement and nondeterminism resolution.



Simple example I

CON1 =

bufin?Value →

set!Value →

get?Stored →

bufout!Stored → *CON1*

MACHINE CELL

VARIABLES *xx*

INVARIANT *xx* : NAT

INITIALISATION *xx* := 00

OPERATIONS

```
  set(yy) = PRE yy : NAT
             THEN xx := yy
             END;
```

```
  yy <-- get = yy := xx
```

END

Translation of Example 1:

Introduce xx

```
#define WORD_TYPE unsigned int
#define WORD_WIDTH 3
WORD_TYPE WORD_WIDTH xx
```

Declare the channels

```
chan WORD_TYPE WORD_WIDTH bufout;
chan WORD_TYPE WORD_WIDTH bufin;
```

The generic
control flow

```
void main(void){
    xx = 0; // initialisation
    SimpleBuffer(bufin, bufout);
}
```

Translating CON1

CON1 =

bufin?Value →

set!Value →

get?Stored →

bufout!Stored → CON1

```
set(yy) = PRE yy : NAT
          THEN xx := yy
          END;
```

```
yy <-- get = yy := xx
```

Translation of CON1

```
macro proc Buffer(bufin, bufout){
WORD_TYPE WORD_WIDTH Stored;
WORD_TYPE WORD_WIDTH Value;
do{
    bufin?Value;    // CSP channel input
    xx = Value;    // body of operation set(Value)
    Stored = xx;   // body of operation Stored <-- get
    bufout!Stored; // CSP channel output
} while(1);
```

Simple example II: parallelism in the controller

$IN = \text{bufin?Value} \rightarrow$
 $\quad \text{set!Value} \rightarrow IN$

$OUT = \text{get?Stored} \rightarrow$
 $\quad \text{bufout!Stored} \rightarrow OUT$

$CON2 = IN \parallel OUT$

```
MACHINE CELL
VARIABLES xx
INVARIANT xx : NAT
INITIALISATION xx := 00
OPERATIONS
  set(yy) = PRE yy : NAT
            THEN xx := yy
            END;

  yy <-- get = yy := xx
END
```

Translation of CON2

CON2

```
macro proc InterleaveBuffer(bufin, bufout){  
  WORD_TYPE WORD_WIDTH Stored;  
  WORD_TYPE WORD_WIDTH Value;  
  par{  
    do {bufin?Value;  
      xx = Value;} while(1);  
  
    do{Stored = xx;  
      bufout!Stored;} while(1);  
  }  
}
```

IN

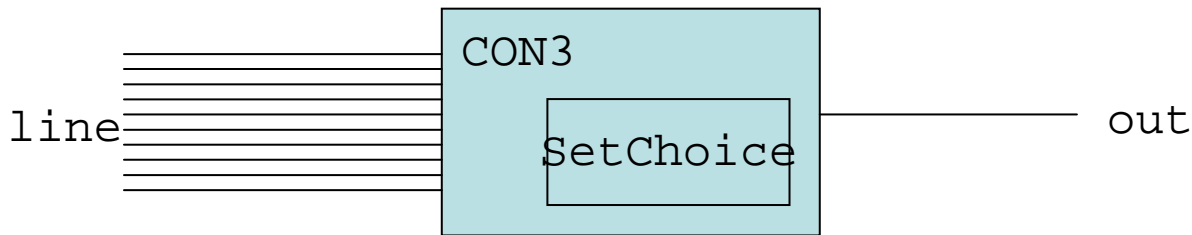
OUT



- Need to make the implemented state concrete (i.e. retrenchment).
- Need to declare and scope all local variables explicitly.
- Preconditions are dropped at implementation.
 - Once consistency is shown at the abstract level, the preconditions have been discharged and are no longer required – they do not appear in implementations.
- Parallel processes proceed in lockstep. In CON2, this yields buffer-like behaviour (though initially non-empty). The Handel-C timing model provides one way of implementing interleaving.
- Natural translation of channel communication and assignment.



Simple Arbiter Example



Abstract data structure
Nondeterminism

CON3 =

$(\parallel_{x:0..15} \text{Read}(x));$

*choose?**y* \rightarrow

*out!**y* \rightarrow *CON3*

Read(x) =

*line.x?**b* \rightarrow

if (*b* = 1)

*then add!**x* \rightarrow *SKIP*

else SKIP

MACHINE *SetChoice*

VARIABLES *ss*

INVARIANT *ss* <: 0..15

INITIALISATION *ss* := {0}

OPERATIONS

yy <-- *choose* =

BEGIN *yy* :: *ss*

|| *ss* := {0}

END;

add(xx) = PRE *xx* : 0..15

THEN *ss* := *ss* \ / {*xx*}

END

END

Refining SetChoice

- Data refinement: set `ss` implemented by array `arr`.
- Resolving underspecification: choice resolved (arbitrarily) by taking the maximum. More complex choice mechanisms possible.
- These refinements are proven correct in B.

```
MACHINE SetChoiceR
VARIABLES arr
INVARIANT arr : 0..15 --> 0..1
           & ss = arr~[ {1} ] & 0 | -> 1 : arr
INITIALISATION arr := (1..15 * {0}) \ / {0 | -> 1}
OPERATIONS
  yy <-- choose =
    BEGIN yy := max(arr~[ {1} ])
           || arr := (1..15 * {0}) \ / {0 | -> 1}
    END;

  add(xx) = arr(xx) := 1

END
```

Translation of SetChoiceR

```
unsigned int 1 arr[16]
```

```
void Init() {  
    par{  
        arr[0] = 1;  
        par (i=1; i<16; i++) {arr[i] = 0; }  
    }  
}
```

```
macro proc add(unsigned int 4 xx){arr[xx] = 1; }
```

```
void choose(unsigned int* yy) {  
    par { max(yy);  
        Init();  
    }  
}
```



Translation of CON3

```
void main() {
    unsigned int 4 y;
    Init();
    par {
        do {
            par (i=0; i<16; i++) {Read(i);}
            choose(&y);
            out!y;
        } while(1);
    }

    macro proc Read(x) {
        unsigned int 1 b;
        line[x]?b; if (b) {add(x); }
        else {delay; }
    }
}
```

CON3 =

$(\parallel_{x:0..15} \text{Read}(x);$
choose?y →
out!y → *CON3*

Read(x) =

line.x?b →
if (b = 1)
then add!x → *SKIP*
else SKIP

- Need to refine closer to implementation before translating.
 - Implementatable data structures.
 - Remove nondeterministic choice.
- Function declaration creates the hardware once (efficient but cannot support concurrent calls) e.g. Init().
- Macro declaration creates hardware once for each call in the code (robust but possibly wasteful) e.g. read, add.
- Handel-C supports reference parameters. Thus B operations returning multiple values can be translated.
- Timing in different branches of an operation.
- Signals vs channels, as implementations for CSP channels.



Subsequent developments



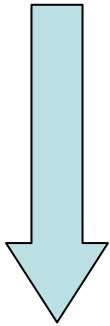
Subsequent Developments

- Development of a larger case study within the Future Technologies project.
 - Development of a 'translation-ready CSP||B' subset, closer to Handel-C.
 - Formal relationship between high level CSP||B models and TR-CSP||B descriptions, within the same semantic framework.
 - Development of a modelling style suited to this approach.
 - Clocked style for CSP||B models, with core functionality of the machine bound up within a single operation.
 - Aim to keep the complexity within the B description, to retain ability to do analysis and verification. Keep the CSP control as simple as possible.



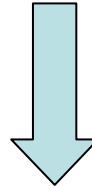
Development path: CSP

CSP||B



Translation
ready
CSP||B

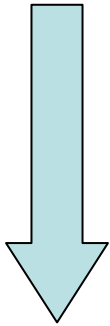
```
CYCLE = load?xx -> CYCLE
[] reset -> CYCLE
[] read -> READ
[] delay -> CYCLE
[] isfull?x?y ->
  if (x == ack_full)
  then RUN
  else CYCLE
```



```
CYCLE(i) = (body!COM_LOAD?v?w?x?y?z ->
            if (i+1 < capacity) then CYCLE(i+1) else CYCLE(capacity))
[] body!COM_RESET?v?w?x?y?z -> CYCLE(0)
[] body!COM_READ?v?w?x?y?z -> READ(0,i)
[] body!COM_NOCOMMAND?v?w?x?y?z -> CYCLE(i)
[] (body!COM_ISFULL?v?w?x?y?z ->
    if (x == ACK_FULL)
    then RUN(i)
    else CYCLE(i))
```

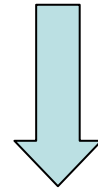
Development path: B

CSP||B



Translation
ready
CSP||B

```
load(ee) =  
  PRE ee : {0, 1}  
  THEN IF size(store) < capacity  
    THEN store := store <- ee  
    ELSE skip  
    END  
  || state := live  
END;
```



ack, outputdata, drive <-- body(command, data, sync) = ...

```
CASE...  
... COM_LOAD THEN  
  local_drivecommand := DRV_UNENABLED  
  || local_outputdata := DAT_NODATA  
  || local_ack := ACK_LOADACK  
  || IF (local_fillcounter = SWITCH_CODE_LENGTH)  
    THEN skip  
    ELSE local_store(local_fillcounter) := data  
      || local_fillcounter := local_fillcounter + 1  
  END
```



Development path: Code

```
switch(command){  
    ...  
    case COM_LOAD:  
    par{  
        local_drivecommand = DRV_UNENABLED;  
        local_outputdata    = DAT_NODATA;  
        local_ack           = ACK_LOADACK;  
        if(local_fillcounter == SWITCH_CODE_LENGTH)  
            delay;  
        else{  
            local_store[local_fillcounter] = data;  
            local_fillcounter++;  
        }  
    }  
    break;
```

Achievements

- Hardware components behaved as intended first time, due to understanding gained from modelling.
- Hardware implementation worked first time with new scenarios.
- Demonstrated an ability to capture abstract behaviour in CSP||B and translate to Handel-C.
- Analysis of abstract platform-independent models.
- Translation to code in a traceable way.
- Translation ready style of CSP||B for translation to Handel-C.



Further considerations

- Handel-C and translation-ready CSP||B affect each other – influence in both directions.
- Need to investigate interactions *between* hardware components.
- TR-CSP||B to Handel-C translation currently done by hand.
- Need a more complete treatment of the translation before we could automate it.
- Interface and timing refinement needs formal justification.

