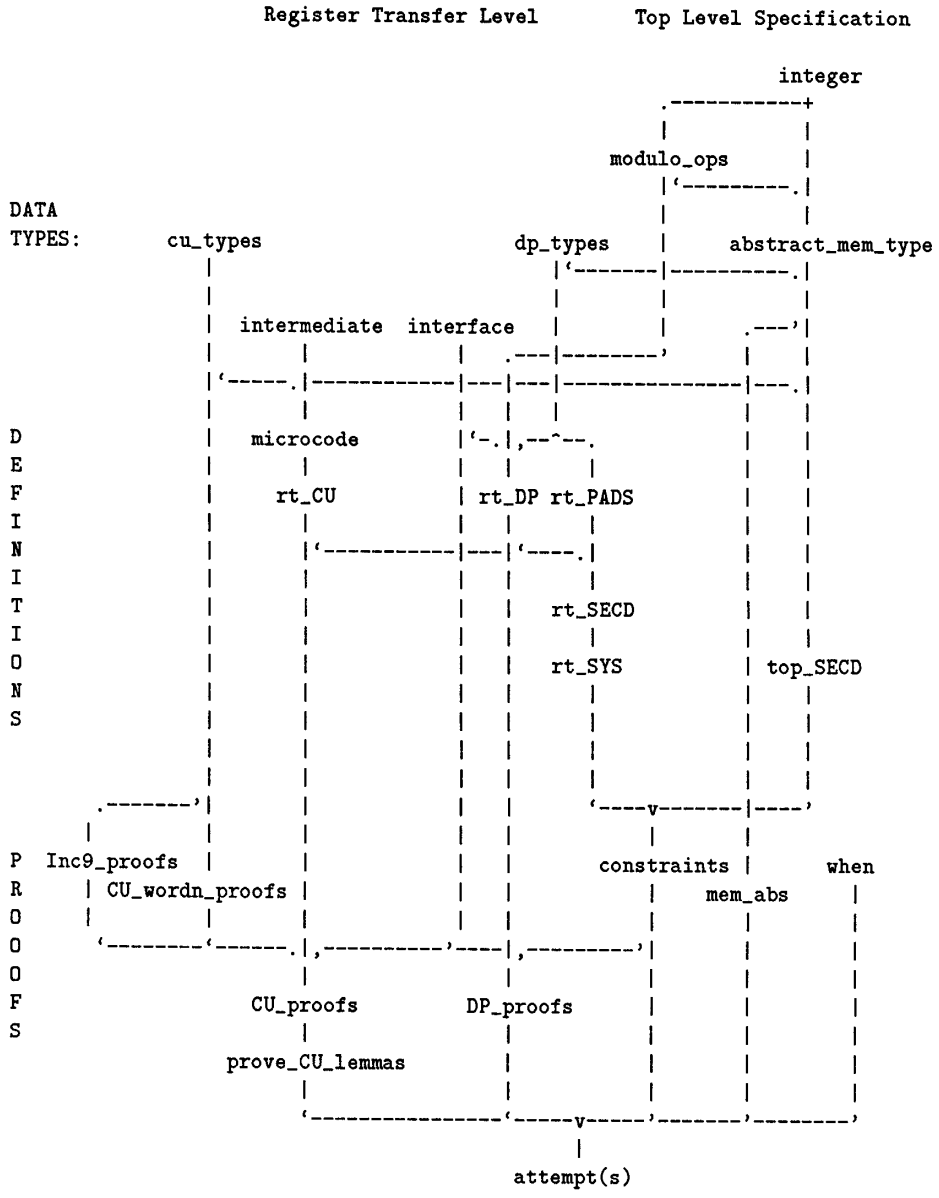
Chapter 1

Introduction

This report covers work in progress on the specification and verification of an SECD chip, designed at the University of Calgary [Gra89]. The hierarchy of theories containing the definitions is summarised in the following dependency diagram. As this work is still in progress, some changes are expected to be made to this material.


Register Transfer Level
Top Level Specification
integer
modulo_ops
DATA
TYPES:
cu_types
dp_types
abstract_mem_type
intermediate
interface
microcode
rt_CU
rt_DP
rt_PADS
rt_SECD
rt_SYS
top_SECD
DEFINITIONS
Inc9_proofs
CU_wordn_proofs
CU_proofs
DP_proofs
prove_CU_lemmas
constraints
mem_abs
when
PROOFS
attempt(s)

Chapter 2

Abstract Data Types

This chapter defines the abstract data types used in the definition of the SECD chip, and its top level behaviour. The major type of concern here is that used to represent finite size words. A package defining fixed size words, called wordn, defines the types of words of lengths 2, 4, 5, 9, 14, 28, and 32, called types ":word2", ":word4", and so on. These are defined in terms of buses of the appropriate width, where a bus is a recursive type defined in terms of constructors "Wire" and "Bus", and can be viewed as simply non-empty lists, with the "Wire" constructor used for the base case.

The top level specification is defined in terms of operations on an abstract memory type.

2.1 RTL Data Types

Data types are separated into those used in the control unit, and those used in the datapath.

2.1.1 cu_types

```
% SECD verification                                                    %
%                                                                     %
% FILE:         cu_types.ml                                           %
%                                                                     %
% DESCRIPTION:  Declares the wordn types used by the control unit.%
%               Also defines field selectors, and an increment        %
%               operation.                                            %
%                                                                     %
% USES FILES:   load_wordn.ml                                         %
%                                                                     %
% Brian Graham 88.04.21                                               %
%                                                                     %
% Modifications:                                                      %
%                                                                     %
%=====================================================================%
new_theory `cu_types`;;

loadf `load_wordn`;;

% ================================================================== %
map (declare_wordn_type true)
     [ 4; % test field of instruction %
```

```
       5; % read, write fields %
       9; % mpc width %
       27 % instruction width %
     ];;

% ================================================================ %
% Field selector functions:                                        %
% Note that wordn maps onto :(bool)bus, not :(num->bool)bus.       %
% ================================================================ %
let w27 =
"Word27 (Bus b27 (Bus b26 (Bus b25 (Bus b24 (Bus b23 (Bus b22
        (Bus b21 (Bus b20 (Bus b19 (Bus b18 (Bus b17 (Bus b16
        (Bus b15 (Bus b14 (Bus b13 (Bus b12 (Bus b11 (Bus b10
        (Bus b9  (Bus b8  (Bus b7  (Bus b6  (Bus b5  (Bus b4
        (Bus b3  (Bus b2  (Wire (b1:bool))))))))))))))))))))))))))))";;

let Word27 = theorem `-` `Word27`;;

let A_field = new_recursive_definition false Word27 `A_field`
   "A_field ^w27 =
      Word9 (Bus b27 (Bus b26 (Bus b25 (Bus b24 (Bus b23 (Bus b22
            (Bus b21 (Bus b20 (Wire b19)))))))))"
   ;;

let Write_field = new_recursive_definition false Word27 `Write_field`
   "Write_field ^w27 =
      Word5 (Bus b5 (Bus b4 (Bus b3 (Bus b2 (Wire b1)))))"
   ;;

let Read_field = new_recursive_definition false Word27 `Read_field`
   "Read_field ^w27 =
      Word5 (Bus b10 (Bus b9 (Bus b8 (Bus b7 (Wire b6)))))"
 ;;

let Alu_field = new_recursive_definition false Word27 `Alu_field`
   "Alu_field ^w27 =
      Word4 (Bus b14 (Bus b13 (Bus b12 (Wire b11))))"
   ;;

let Test_field = new_recursive_definition false Word27
   `Test_field`
   "Test_field ^w27 =
      Word4 (Bus b18 (Bus b17 (Bus b16 (Wire b15))))"
   ;;

% ================================================================ %
% Define an increment function on objects of type :word9.          %
% This is done by tail recursion on buses, with the Inc_9          %
% function converting to from and back to the :word9 type,         %
% as well as selecting the bus returned by inc.                    %
%                                                                  %
% This is an implementation sort of definition, but is sensible    %
% here since the "meaning" is not important, only the value, as    %
% an argument to the ROM defintion.                                %
```

```
% ================================================================ %
let inc = new_recursive_definition false (theorem `bus` `bus_axiom`)
   `inc`
  "(inc (Wire b)    = (Wire (~b),b))  /\
   (inc (Bus b bus) = (let (bs,fg) = (inc bus)
                       in
                      (Bus (fg => ~b | b) bs, (fg /\ b)))
   )";;

let Inc9 = new_definition
  (`Inc9`,
   "Inc9 = Word9 o FST o inc o Bits9"
  );;

close_theory ();;
print_theory `-`;;
```

2.1.2 dp_types

```
% SECD verification                                                    %
%                                                                     %
% FILE:         dp_types.ml                                           %
%                                                                     %
% DESCRIPTION:  Declares the wordn types used by the datapath.        %
%               Also defines field selectors, and various data        %
%               type related constants and functions.                 %
%                                                                     %
% USES FILES:   load_wordn.ml, integer library                        %
%                                                                     %
% Brian Graham 88.04.21                                               %
%                                                                     %
% Modifications:                                                      %
%                                                                     %
%=====================================================================%
new_theory `dp_types`;;

loadf `load_wordn`;;
load_library `integer`;;

map (declare_wordn_type true)
     [ 2;  % garbage bits and type bits of record %
       14; % pointer size %
       28; % number size %
       32  % record size %
     ];;

let NIL_addr  = new_definition
                (`NIL_addr`,   "NIL_addr  = #00000000000000")
and NUM_addr  = new_definition
                (`NUM_addr`,   "NUM_addr  = #11111111111111")
and T_addr    = new_definition
                (`T_addr`,     "T_addr    = #00000000000001")
and F_addr    = new_definition
                (`F_addr`,     "F_addr    = #00000000000010");;

let ZEROS14   = new_definition
                (`ZEROS14`,    "ZEROS14   = #00000000000000")
and ZERO28    = new_definition
                (`ZERO28`,     "ZERO28    = #0000000000000000000000000000");;

let RT_SYMBOL = new_definition
                (`RT_SYMBOL`,  "RT_SYMBOL = #10")
and RT_NUMBER = new_definition
                (`RT_NUMBER`,  "RT_NUMBER = #11")
and RT_CONS   = new_definition
               (`RT_CONS`,     "RT_CONS   = #00");;

% ================ sub-field extractor functions ================ %
% SEG and V are no longer defined in the system.  further, the    %
% definition of EL is reversed from the previous built-in defn.   %
% EL 0 l = hd l  instead of the last element.                     %
```

```
% thus, EL_CONV does weird things, and cannot be used.            %
% ================================================================ %
load_theorem `-` `Word32`;;

let w32 = "Word32 (Bus b32 (Bus b31 (Bus b30 (Bus b29 (Bus b28
                  (Bus b27 (Bus b26 (Bus b25 (Bus b24 (Bus b23
                  (Bus b22 (Bus b21 (Bus b20 (Bus b19 (Bus b18
                  (Bus b17 (Bus b16 (Bus b15 (Bus b14 (Bus b13
                  (Bus b12 (Bus b11 (Bus b10 (Bus b9  (Bus b8
                  (Bus b7  (Bus b6  (Bus b5  (Bus b4  (Bus b3
                  (Bus b2  (Wire (b1:bool)
                  ))))))))))))))))))))))))))))))))";;


let garbage_bits = new_recursive_definition false Word32
   `garbage_bits`
   "garbage_bits ^w32 = Word2 (Bus b32 (Wire b31))";;

let mark_bit = new_recursive_definition false Word32
   `mark_bit`
   "mark_bit ^w32 = (b32:bool)";;

let field_bit = new_recursive_definition false Word32
   `field_bit`
   "field_bit ^w32 = (b31:bool)";;

let rec_type_bits = new_recursive_definition false Word32
   `rec_type_bits`
   "rec_type_bits ^w32 = Word2 (Bus b30 (Wire b29))";;

let car_bits = new_recursive_definition false Word32
   `car_bits`
   "car_bits ^w32 =
    Word14 (Bus b28 (Bus b27 (Bus b26 (Bus b25 (Bus b24 (Bus b23
           (Bus b22 (Bus b21 (Bus b20 (Bus b19 (Bus b18 (Bus b17
           (Bus b16 (Wire b15)))))))))))))))";;

let cdr_bits = new_recursive_definition false Word32
   `cdr_bits`
   "cdr_bits ^w32 =
    Word14 (Bus b14 (Bus b13 (Bus b12 (Bus b11 (Bus b10 (Bus b9
           (Bus b8  (Bus b7  (Bus b6  (Bus b5  (Bus b4  (Bus b3
           (Bus b2  (Wire b1)))))))))))))))";;

let atom_bits = new_recursive_definition false Word32
   `atom_bits`
   "atom_bits ^w32 =
    Word28 (Bus b28 (Bus b27 (Bus b26 (Bus b25 (Bus b24 (Bus b23
           (Bus b22 (Bus b21 (Bus b20 (Bus b19 (Bus b18 (Bus b17
           (Bus b16 (Bus b15 (Bus b14 (Bus b13 (Bus b12 (Bus b11
           (Bus b10 (Bus b9  (Bus b8  (Bus b7  (Bus b6  (Bus b5
           (Bus b4  (Bus b3  (Bus b2  (Wire b1)
                           ))))))))))))))))))))))))))))))";;
```

```
% ================ recognizer functions ========================== %
%    note that these operate only on the record type field         %
% Revised 89.09.07 - they now work on ":word32" arguments.         %
% ================================================================ %
let is_symbol = new_definition
  (`is_symbol`,
   "!x:word32. is_symbol x = (rec_type_bits x = RT_SYMBOL)"
  );;

let is_number = new_definition
  (`is_number`,
   "!x:word32. is_number x = (rec_type_bits x = RT_NUMBER)"
  );;

let is_cons = new_definition
  (`is_cons`,
   "!x:word32. is_cons x = (rec_type_bits x = RT_CONS)"
  );;

let is_atom = new_definition
  (`is_atom`,
   "!x:word32. is_atom x = (is_symbol x \/ is_number x)"
  );;

% ================================================================ %
% Data abstraction functions.                                      %
% ================================================================ %
let bus_axiom = theorem `bus`      `bus_axiom`;;

let bv = new_definition
  (`bv`, "bv x = x => 1 | 0");;

let bv_thm = prove_thm
 ( `bv_thm`,
   "(bv T = 1)  /\  (bv F = 0)",
   REWRITE_TAC [bv]);;

let val = new_recursive_definition false bus_axiom `val`
  "(val n (Wire w)    =  n + n + (bv w)          ) /\
   (val n (Bus b bus) =  val (n + n + (bv b)) bus)
  ";;

let Val = new_definition
  (`Val`, "Val = val 0");;

let iVal = new_recursive_definition false bus_axiom `iVal`
  "(iVal (Wire w)    =  neg (INT (bv w))                  ) /\
   (iVal (Bus b bus) =  INT (val 0 bus)  minus
                        INT ((2 EXP (Width bus)) * bv(b))  )
  ";;

close_theory ();;
print_theory `-`;;
```

2.2 Abstract Memory Type

Both the rtl and top level specifications use modulo integer operations that are declared in a common theory file. The definition of the particular operations is not an issue at this level of proof, but concerns the low level implementation of the ALU. Thus, at this level we need only assure that the selected operation is indeed performed on the correct arguments, and leave the description of the operation to a lower level of specification. Naturally, this will propagate to the top level specification of the chip.

2.2.1 modulo_ops

```
% SECD verification                                                    %
%                                                                     %
% FILE:                modulo_ops.ml                                  %
%                                                                     %
% DESCRIPTION:                                                        %
% This theory merely introduces the constant names for the            %
% modulo 28 arithmetic operations performed by the secd's ALU.        %
% These names are used in both the rt level spec, as well as          %
% the top level spec, and thus for this stage of proof, no            %
% definition for the terms is necessary.                              %
% The decriment operation will need definition, however, for          %
% proof of the LD instruction.                                        %
%                                                                     %
% USES FILES:   integer library                                       %
%                                                                     %
% Brian Graham 89.10.06                                               %
%                                                                     %
% Modifications:                                                      %
%                                                                     %
%=====================================================================%
new_theory `modulo_ops`;;
load_library `integer`;;
%=====================================================================%
% We create new constants for the modulo arithmetic operations       %
% that the machine executes.  The definition of these operations      %
% will propagate through from the lower level definitions:            %
% in proving the alu, we will show that the abstraction to            %
% integers and operations performed on them constitutes a modulo      %
% arithmetic operation.                                               %
%=====================================================================%
new_constant ( `modulo_28_Dec`, ":integer->integer");;

map new_curried_infix [ `modulo_28_Add`, ":integer->integer->integer"
                      ; `modulo_28_Sub`, ":integer->integer->integer"
                      ];;
close_theory ();;
```

2.2.2 abstract_mem_type

```
% SECD verification                                                    %
%                                                                     %
% FILE:         abstract_mem_type.ml                                  %
%                                                                     %
% DESCRIPTION: I am attempting here to define memories as per         %
%              Ian Mason's work on the Semantics of Destructive       %
%              Lisp.                                                  %
%              This differs from the original attempt in that the     %
%              selector functions also have "free" as an argument,    %
%              although they apply the identity to it, never          %
%              changing the free pointer.  This eases composition     %
%              of cons'es and selector functions (car's cdr's).       %
%                                                                     %
%              This third revision alters the definition of the       %
%              memory function type.  This time the codomain of       %
%              the memory function is domain squared or atom.         %
%              This relieves the problem of a different mapping       %
%              for atoms that was implied by ian's work, where        %
%              perhaps the pointer and atom are stored in same        %
%              field width, rather than placing atom in a             %
%              separate cell, as the secd implementation does.        %
%                                                                     %
% Brian Graham 88.08.17                                               %
%                                                                     %
% Modifications                                                       %
% 89.05.25 - converted to hol88 type package                          %
%=====================================================================%

new_theory `abstract_mem_type`;;
loadf `load_wordn`;;
new_parent `modulo_ops`;;

load_library `integer`;;

%=====================================================================%
% General theory of Memory Structures:                                %
% ************************************                                %
% C - the set of cells of memory                                      %
% A - the set of atoms :                                              %
%     includes (a subset of) the integers                             %
%     symbols - including NIL, TRUE, FALSE                            %
% C and A are disjoint.                                               %
%                                                                     %
% V - the set of memory values: V = C U A                             %
%                                                                     %
% mu - a memory, which is a function from finite subset of C          %
%      to the set of sequences of memory values: V* = (A U C)*        %
% (This is a most general definition.  In lisp we typically have      %
%  sequences of length 2 in cons cells.)                              %
%                                                                     %
% delta(mu) = domain of mu                                            %
%                                                                     %
```

```
% mu is in [delta(mu) -> (delta(mu) U A)*]                              %
%                                                                       %
% M(A,C) - the set of all memories over A and C                         %
%                                                                       %
% [v0, ..., vn-1],mu - a memory object, consists of a memory            %
%    mu in M, and a sequence s.t. vi is in (delta(mu) U A)              %
%    (Intuitively, a memory and a set of values which EXIST in          %
%                  that memory).                                        %
%=======================================================================%

%=======================================================================%
% Theory of S-expression Memories:                                      %
% ********************************                                      %
% A - includes Z (the integers), NIL, TRUE, FALSE, and unlimited        %
%     collection of non-numeric atoms.                                  %
%                                                                       %
% mu - is in (delta(mu) -> (A U (delta(mu)^2)))                         %
%                                                                       %
% O = (Car, Cdr, Cons, Rplaca, Rplacd, Atom, Add, Sub, Mul, Dec,        %
%           Div, Rem, Eq, Leq, ...                                      %
% All operations except Cons are obvious.  Cons will enlarge the        %
% domain (delta(mu)) of the memory, by selecting a new location         %
% from free storage, and installing the arguments as its contents.      %
% Free storage is just (C - delta(mu)).                                 %
%                                                                       %
% cons([v0,v1];mu) = c;mu0                                              %
%         where c is not in delta(mu), and                              %
%               mu0 = mu{c<-[v0,v1]}                                    %
%                                                                       %
% Rplaca and Rplacd update the contents of a location in memory,        %
% but do not change the domain.                                         %
%=======================================================================%

%=======================================================================%
% Implementation of S-expression Memories:                              %
% ****************************************                              %
% I want to define a memory as a function from some domain to           %
% (the domain squared Union some other set (atom)).                     %
% Thus it will be a type of two type variables, delta (for domain)      %
% and alpha (for the set of atoms).  Thus we want:                      %
%      delta -> (delta^2 U alpha)                                       %
%                                                                       %
% Now, the same idea for a memory with mark and field bits for          %
% garbage collection: mfsexp.                                           %
%                                                                       %
% For simplicity, a free list pointer will be included as an            %
% argument to all functions on mfsexp_mem's.  Thus, the domain          %
% of the function will not change, as done in the theory above.         %
% Rather, the domain stays constant, and the cells not in the           %
% domain of the theoretical version of the function, will be in         %
% the free list.                                                        %
% This makes the allocation of cells explicit, and this is              %
% really not an essential part of the specification - we don't          %
% care what shape the free list is in, as long as the relevant          %
```

```
% data structures are correctly manipulated.                          %
%                                                                      %
% Memory is mapped onto an existing (function) type, and is           %
% defined in 2 steps:                                                  %
%   1. define a recognizer function on the existing type.              %
%   2. show the type is nonempty.                                      %
%======================================================================%
let IS_mfsexp_mem = new_definition
  (`IS_mfsexp_mem`,
   "IS_mfsexp_mem (rep_mfsexpmem:*->((bool#bool)#((*#*)+**))) =
    !cell:*. ?m f.
             (?a d:*. rep_mfsexpmem cell = (m,f),INL(a,d)) \/
             (? z:**. rep_mfsexpmem cell = (m,f),INR z)"
  );;

let EXISTS_mfsexp_mem = prove_thm
  (`EXISTS_mfsexp_mem`,
   "?m:*->((bool#bool)#((*#*)+**)). IS_mfsexp_mem m",
   EXISTS_TAC "(\a:*. ((F,F),(INL (a,a)):(*#*)+**))"
   THEN port [IS_mfsexp_mem]
   THEN in_conv_tac BETA_CONV
   THEN GEN_TAC
   THEN REPEAT (EXISTS_TAC "F")
   THEN DISJ1_TAC
   THEN REPEAT (EXISTS_TAC "(cell):*")
   THEN MATCH_ACCEPT_TAC (REFL "x:*")
  );;

%======================================================================%
% mfsexp_mem_thm = |- ?rep. TYPE_DEFINITION IS_mfsexp_mem rep         %
%======================================================================%
let mfsexp_mem_thm = new_type_definition
  (`mfsexp_mem`,
   "IS_mfsexp_mem:(* -> ((bool#bool)#((*#*)+**)))->bool",
   EXISTS_mfsexp_mem
  );;

%======================================================================%
% Get the set of lemmas that Tom Melham's type package provides.      %
%======================================================================%
loadt `wordn/new_ty_lem`;;

map2 save_thm
     ( [ `REP_mfsexp_mem_11`
       ; `REP_mfsexp_mem_ONTO`
       ; `ABS_mfsexp_mem_11`
       ; `ABS_mfsexp_mem_ONTO`
       ; `ABS_mfsexp_mem_REP_mfsexp_mem`
       ; `REP_mfsexp_mem_ABS_mfsexp_mem`
       ] , prove_new_type_lemmas mfsexp_mem_thm );;
```

```
%===================================================================%
% The type of memory that we want has:                              %
%       domain:     :word14                                         %
%       range:      :(bool # bool) # ((word14 # word14) + atoms)    %
% The set of atoms includes:                                        %
%       integers                                                    %
%       symbols                                                     %
% Among the symbols are the constants: NIL, TRUE, and FALSE.        %
% (Note that the special values for TRUE and FALSE are not          %
%  typical for Lisp implementations, but are used by LispKit.       %
%  These values are used by 4 machine instructions:                 %
%       SEL, ATOM, EQ, and LEQ.)                                    %
%                                                                   %
% Symbols other than those listed above can occur as arguments to   %
% the compiled program.  In the machine (and the actual memory)     %
% they are represented by numbers.  Program variables, on the       %
% other hand, are not directly represented in the memory, but       %
% instead, references to the variable are compiled into a LD        %
% instruction with arguments that enable it to locate the           %
% position where the value bound to the variable is stored.  Thus   %
% variables are not represented in the set of atoms.                %
%                                                                   %
% It may not be useful to define a new type for atoms, instead      %
% of using the type ":integer+num" (?) alone.                       %
%===================================================================%

%===================================================================%
% atom_thm =                                                        %
% |- !f0 f1. ?! fn. (!i. fn(Int i) = f0 i) /\                       %
%                   (!n. fn(Symb n) = f1 n)                         %
%===================================================================%
let atom_thm = define_type
   `atom_thm`
   `atom = Int integer | Symb num`;;

%===================================================================%
% |- !P. (!i. P(Int i)) /\ (!n. P(Symb n)) ==> (!a. P a)            %
%===================================================================%
let atom_Induct = save_thm
  (`atom_Induct`, prove_induction_thm atom_thm);;

%===================================================================%
% |- !a. (?i. a = Int i) \/ (?n. a = Symb n)                        %
%===================================================================%
let atom_cases = save_thm
  (`atom_cases`, prove_cases_thm atom_Induct);;

%===================================================================%
% Note that the functions:                                          %
%       prove_constructors_one_one and                              %
%       prove_constructors_distinct                                 %
% fail with atom_Induct.                                            %
%===================================================================%
```

```
let int_of_atom = new_recursive_definition false atom_thm
   `int_of_atom`
   "int_of_atom (Int i) = i";;

let symb_of_atom = new_recursive_definition false atom_thm
   `symb_of_atom`
   "symb_of_atom (Symb s) = s";;

let atom_is_int = new_recursive_definition false atom_thm
   `atom_is_int`
  "(atom_is_int (Symb s) = F) /\
   (atom_is_int (Int  i) = T)   ";;

let atom_is_symb = new_recursive_definition false atom_thm
   `atom_is_symb`
  "(atom_is_symb (Symb s) = T) /\
   (atom_is_symb (Int  i) = F)   ";;

%=====================================================================%
% Built-in constants:                                                 %
%=====================================================================%
let NIL_atom = new_definition (`NIL_atom`,"NIL_atom:atom = Symb 0")
and T_atom   = new_definition (`T_atom`,  "T_atom  :atom = Symb 1")
and F_atom   = new_definition (`F_atom`,  "F_atom  :atom = Symb 2");;


%=====================================================================%
% M_Car:(C#M#C) -> C                                                  %
%=====================================================================%
let M_Car = new_definition
  (`M_Car`,
   "M_Car (x:*, MEM:(*,**)mfsexp_mem,free:*) =
            FST (OUTL (SND (REP_mfsexp_mem MEM x)))"
  );;

%=====================================================================%
% M_CAR:(C#M#C) -> (C#M#C)                                            %
%=====================================================================%
let M_CAR = new_definition
  (`M_CAR`,
   "M_CAR (x, MEM:(*,**)mfsexp_mem,free:*) =
            M_Car (x, MEM, free), MEM, free"
  );;

%=====================================================================%
% M_Cdr:(C#M#C) -> C                                                  %
%=====================================================================%
let M_Cdr = new_definition
  (`M_Cdr`,
   "M_Cdr (x:*, MEM:(*,**)mfsexp_mem,free:*) =
            SND (OUTL (SND (REP_mfsexp_mem MEM x)))"
  );;
```

```
%=====================================================================%
% M_CDR:(C#M#C) -> (C#M#C)                                            %
%=====================================================================%
let M_CDR = new_definition
  (`M_CDR`,
   "M_CDR (x, MEM:(*,**)mfsexp_mem,free:*) =
           M_Cdr (x, MEM, free), MEM, free"
  );;

%=====================================================================%
% M_Atom:(C#M#C) -> bool                                              %
%=====================================================================%
let M_Atom = new_definition
  (`M_Atom`,
   "!(v:*) (MEM:(*,**)mfsexp_mem) (free:*).
    M_Atom (v,MEM,free) = ISR(SND(REP_mfsexp_mem MEM v))"
  );;


%=====================================================================%
% This definition will not suffice for the next stage when            %
% garbage collection is attempted.  Garbage collection is             %
% performed at a lower level, protecting structures accessible        %
% from various registers not seen at this level.  The function        %
% should have a set of pointers as arguments, aside from the          %
% free list pointer.                                                  %
%=====================================================================%
new_constant
(`M_garbage_collect`, ":((*,**)mfsexp_mem#*)->((*,**)mfsexp_mem#*)");;

%=====================================================================%
% This version allows composition of Cons's, with one added           %
% argument each time.  Keeps track of the state of memory and         %
% free pointer nicely as well!                                        %
%                                                                     %
% M_Cons:(C#C#M#C) -> (C#M#C)                                         %
%=====================================================================%
let M_Cons = new_definition
  (`M_Cons`,
   "M_Cons (x:*, y:*, MEM:(*,**)mfsexp_mem, free:*) =
         let (Mem,Free) = (M_Atom (free,MEM,free)         =>
                           (M_garbage_collect (MEM, free)) |
                           (MEM, free))
         in
         ( Free,
           (@m. REP_mfsexp_mem m =
             (\a. (a=Free) => (F,F),INL(x,y) |
                              REP_mfsexp_mem Mem a)),
           M_Cdr (Free, Mem, Free)
         )"
  );;
```

```
%=====================================================================%
% Transposes order of arguments only.  Useful for composing.         %
%=====================================================================%
let M_Cons_tr = new_definition
  (`M_Cons_tr`,
   "M_Cons_tr (x:*,y:*, MEM:(*,**)mfsexp_mem, free:*) =
         M_Cons (y,x,MEM,free)"
  );;

%=====================================================================%
% Very similar to the definition of a new function as done in         %
% M_Cons, the destructive replace operations are simpler.             %
%=====================================================================%
%=====================================================================%
% M_Rplaca:(C#C#M#C) -> (C#M#C)                                       %
%=====================================================================%
let M_Rplaca = new_definition
  (`M_Rplaca`,
   "M_Rplaca (c:*,v:*, MEM:(*,**)mfsexp_mem,free:*) =
        ( c,
          (@m. REP_mfsexp_mem m =
             (\a:*. (a = c) => (F,F),INL(v, M_Cdr(a, MEM, free)) |
                               REP_mfsexp_mem MEM a)),
          free
        )"
  );;
%=====================================================================%
% M_Rplacd:(C#C#M#C) -> (C#M#C)                                       %
%=====================================================================%
let M_Rplacd = new_definition
  (`M_Rplacd`,
   "M_Rplacd (c:*,v:*, MEM:(*,**)mfsexp_mem,free:*) =
        ( c,
          (@m. REP_mfsexp_mem m =
             (\a:*. (a = c) => (F,F),INL(M_Car(a, MEM, free),v) |
                               REP_mfsexp_mem MEM a)),
          free
        )"
  );;

%=====================================================================%
% The functions to get the values of the fields is fairly simple,     %
% but should also be defined for constants!  Here they are not.       %
% This is not a vital issue, until garbage collection is tackled.     %
%                                                                     %
% The comment above doesn't seem correct!!!                           %
%=====================================================================%
let M_mark = new_definition
  (`M_mark`,
   "M_mark (p:*) (MEM:(*,**)mfsexp_mem) =
     (FST o FST) ((REP_mfsexp_mem MEM) p)"
  );;
```

```
let M_field = new_definition
  ('M_field',
   "M_field (p:*) (MEM:(*,**)mfsexp_mem) =
     SND (FST ((REP_mfsexp_mem MEM) p))"
  );;

let M_stripped = new_definition
  ('M_stripped',
   "M_stripped (p:*) (MEM:(*,**)mfsexp_mem) =
     SND ((REP_mfsexp_mem MEM) p)"
  );;
%=====================================================================%
% The functions to set the gc bits are similar to the destructive     %
% replace operations.                                                 %
%=====================================================================%
let M_setm = new_definition
  ('M_setm',
   "M_setm (b:bool, c:*, MEM:(*,**)mfsexp_mem,free:*) =
            (@ m. REP_mfsexp_mem m =
              \a. (a = c) =>
                   (b,M_field c MEM), (M_stripped c MEM) |
                   REP_mfsexp_mem MEM a)"
  );;

let M_setf = new_definition
  ('M_setf',
   "M_setf (b:bool, c:*, MEM:(*,**)mfsexp_mem, free:*) =
            (@ m. REP_mfsexp_mem m =
              \a. (a = c) =>
                   (M_mark c MEM,b), (M_stripped c MEM) |
                   REP_mfsexp_mem MEM a)"
  );;

%=====================================================================%
% There are only a few remaining operations to define.               %
% These are :                                                         %
%  int (???)                                                          %
%  dec                                                                %
%  eq, leq                                                            %
% add, sub                                                            %
% (mul, div, rem) ???                                                 %
% These may or may not be necessary for what we need to do.          %
%=====================================================================%

%=====================================================================%
% First some useful selectors and predicates.                         %
%=====================================================================%
let M_atom_of = new_definition
  ('M_atom_of',
   "!(v:*) (MEM:(*,atom)mfsexp_mem) (free:*).
    M_atom_of (v,MEM,free) = OUTR(SND(REP_mfsexp_mem MEM v))"
  );;
```

```
let M_int_of = new_definition
  (`M_int_of`,
   "!(v:*) (MEM:(*,atom)mfsexp_mem) (free:*).
    M_int_of (v,MEM,free) = int_of_atom (M_atom_of (v,MEM,free))"
  );;

% Omit unless needed ... this is complex ...
let Is_int = new_definition
  (`Is_int`,
   "!(v:*) (MEM:(*,atom)mfsexp_mem) (free:*).
    Is_int (v,MEM,free) =
         ISL(REP_atom(OUTR(SND(REP_mfsexp_mem MEM v))))"
  );;
%

let M_is_T = new_definition
  (`M_is_T`,
   "!(v:*) (MEM:(*,atom)mfsexp_mem) (free:*).
    M_is_T (v,MEM,free) = (M_atom_of (v,MEM,free) = T_atom)"
  );;

%=====================================================================%
% New int values may be produced and stored in the memory, BUT,       %
% it is impossible to create new symbols!  Thus we never store        %
% a Symbol atom in memory.  Instead, the symbols are used in          %
% data structures that may be built up by using pointers to the       %
% symbols, wherever they are located.                                 %
% The 3 symbolic constants are in reserved memory locations:          %
% this is a constraint on the memory state.  This permits using       %
% the address of the constant locations, instead of the constants     %
% themselves!                                                         %
%=====================================================================%
let M_store_int = new_definition
  (`M_store_int`,
   "M_store_int (i:integer, MEM:(*,atom)mfsexp_mem,free:*) =
     let (Mem,Free) = M_Atom (free,MEM,free)        =>
                      M_garbage_collect (MEM,free) |
                      (MEM,free)
     in
     ( Free,
       (@m. REP_mfsexp_mem m =
            (\a. (a = Free) => (F,F),INR (Int i) |
                               REP_mfsexp_mem Mem a)),
       M_Cdr (Free,Mem,Free)
     )"
  );;
```

```
%================================================================%
% Now, the required functions.                                   %
%================================================================%
let M_Dec = new_definition
  (`M_Dec`,
   "M_Dec (v:*,MEM:(*,atom)mfsexp_mem,free:*)  =
      M_store_int (modulo_28_Dec (M_int_of (v,MEM,free)), MEM, free)"
  );;

%================================================================%
% Note that EQ is only defined for atoms:                        %
%       at least one of the arg's must be an atom                %
% The default case when neither are atoms returns an expression  %
% which cannot be evaluated.  This attempts to approximate       %
% undefinedness.                                                 %
%                                                                %
% Revised 89.09.06 - the definition more closely approximates the %
%                    style of definition used by the rt level in %
%                    the flagsunit.                              %
%================================================================%
let M_Eq = new_definition
  (`M_Eq`,
   "M_Eq (x:*,y:*,MEM:(*,atom)mfsexp_mem,free:*)  =
      (M_Atom (x,MEM,free) \/ M_Atom (y,MEM,free)) =>
        (M_Atom (x,MEM,free) /\
         M_Atom (y,MEM,free) /\
         (M_atom_of (x,MEM,free) = M_atom_of (y,MEM,free)))
                                                        |
      @z.F"
   );;

let M_Leq = new_definition
  (`M_Leq`,
   "M_Leq (x:*,y:*,MEM:(*,atom)mfsexp_mem,free:*) =
        let x_val = M_int_of (x,MEM,free)
        in
        let y_val = M_int_of (y,MEM,free)
        in
        ((x_val below y_val) \/ (x_val = y_val))"
  );;

let M_Add = new_definition
  (`M_Add`,
   "M_Add (x:*,y:*,MEM:(*,atom)mfsexp_mem,free:*) =
        let x_val = M_int_of (x,MEM,free)
        in
        let y_val = M_int_of (y,MEM,free)
        in
        M_store_int (x_val modulo_28_Add y_val, MEM, free)"
  );;
```

```
let M_Sub = new_definition
  ('M_Sub',
   "M_Sub (x:*,y:*,MEM:(*,atom)mfsexp_mem,free:*) =
        let x_val = M_int_of (x,MEM,free)
        in
        let y_val = M_int_of (y,MEM,free)
        in
        M_store_int (x_val modulo_28_Sub y_val, MEM, free)"
  );;

close_theory ();;

print_theory `-`;;
```

Chapter 3

Definitions

3.1 RTL Definition

3.1.1 microcode

```
% SECD verification                                                    %
%                                                                     %
% FILE:         microcode.ml                                          %
%                                                                     %
% DESCRIPTION: This file reads the intermediate form of the           %
%              secd microcode and defines the ROM device.             %
%                                                                     %
% USES FILES:  cu_types                                               %
%                                                                     %
% Brian Graham 88.04.21                                               %
%                                                                     %
% Modifications:                                                      %
% 88.11.09 - removed the definitions for match, rom, ROM             %
% 89.07.17 - ISD: added theorems for ROM                              %
% 89.09.01 - BtG: Changed to a definition of ROM_fun from ROM.        %
%                 This makes unwinding easier for proof of CU.        %
%                                                                     %
%=====================================================================%
new_theory `microcode`;;

loadf `load_wordn`;;

new_parent `cu_types`;;

% ============================================================ %
% I/O routines:                                                %
% *************                                                %
% The ml io functions are a bit too limited.  This io stuff    %
% permits reading int's from a file, rather than just          %
% individual characters.                                       %
% ============================================================ %
let lnil = obj_of_string `nil`
and lquote = obj_of_string `quote`;;
```

```
let ratom port =
  let a = paired_cons ((obj_of_string 'EOF'),lnil) in
  let b = paired_cons (lquote,a) in
  let c = paired_cons (b,lnil) in
  let d = paired_cons (port,c) in
  let e = paired_cons((obj_of_string 'ratom'),d) in
  let f = lisp_eval e in
  (is_int f) => (int_of_obj f) | failwith 'end-of-file';;

% ============================================================ %
% open the input file ...                                      %
% ============================================================ %
let in_file_string = openi 'intermediate';;
let in_file = obj_of_string in_file_string;;

% ============================================================ %
% Processing routines for the input:                           %
% *********************************                            %
% ============================================================ %

% ============================================================ %
% Convert a num to a boolean list (1's and 0's).               %
%                                                              %
% Functions:                                                   %
%   mk_bit_list (num,size) res                                 %
%       returns a list of strings of 1's and 0's,              %
%       of length size.                                        %
%   example : #mk_bit_list (3,4)[];;                           %
%             ['0'; '0'; '1'; '1'] : string list               %
%                                                              %
%   mk_bit_list_list pairl                                     %
%       returns a list of all the string representations of    %
%       the number,size pairs, appended into one list.         %
%       Note that only CONS is used, not append.               %
%       Also, the list keeps the same order.                   %
%   example : mk_bit_list_list [3,3;2,3];;                     %
%             ['0'; '1'; '1'; '0'; '1'; '0'] : string list     %
% ============================================================ %
let rem (x,y) = x - y * (x / y);;

letrec mk_bit_list (num, size) res =
   (size = 0) => res |
                 (mk_bit_list ((num/2), (size-1))
                   (((string_of_int o rem) (num,2)) . res));;

letrec mk_bit_list_list pairl =
   (pairl = []) => [] |
                  mk_bit_list (hd pairl)
                   (mk_bit_list_list (tl pairl));;
```

```
% ============================================================ %
% Two functions for creating constants.                        %
% These 2 functions return the appropriate wordn object, from  %
% a pair or list of pairs.                                     %
% ============================================================ %
let mk_word9_from_num n =
        mk_const
           (implode (`#` . (mk_bit_list (n,9) ([]:(string)list))),
           ":word9");;

let mk_word27_from_list l =
        mk_const (implode (`#` . (mk_bit_list_list l)), ":word27");;

% ============================================================ %
% Process one address: read in the microcode intermediate      %
% form, and convert it to a pair: (address, code):word9#word27.%
% ============================================================ %
let mk_micro_instr in_file =

  let addr = ratom in_file
  and status = ratom in_file
  and read = ratom in_file
  and write = ratom in_file
  and alu = ratom in_file
  and test = ratom in_file
  and A = ratom in_file                                    in

  let field_list = [A,9; test,4; alu,4; write,5; read,5] in

 (mk_word9_from_num addr, mk_word27_from_list field_list) ;;

% ============================================================ %
% This is used only by the following recursive function.       %
%                                                              %
% letrec make_microcode_list n in_file =                       %
%      (n < 0) => [],[] |                                      %
%                 (join (mk_micro_instr in_file)               %
%                       (make_microcode_list (n-1) in_file))   %
% where join (a,b) (c,d) = (a.c, b.d)                          %
% ;;                                                           %
% ============================================================ %
```

```
% ========================================================== %
% Note that this function relies upon eager evaluation of    %
%      make_microcode_list last_addr in_file                 %
% in order to close the file after, and still return the     %
% result in normal function style.                           %
%                                                            %
% However, the function overflows the NAMESTACK!!!           %
%                                                            %
% let load_microcode_list n =                                %
% let last_addr = (ratom in_file) - n                        %
% in                                                         %
% let result = make_microcode_list last_addr in_file         %
% in                                                         %
% close in_file                                              %
% ;                                                          %
% result;;                                                   %
% ========================================================== %
letrec make_microcode_list n in_file =
     (n = 0) => [],[] |
                (join (mk_micro_instr in_file)
                      (make_microcode_list (n-1) in_file))
 where join (a,b) (c,d) = (a.c, b.d)
;;

let load_microcode_list n =
 let last_addr = (ratom in_file) - n
 in
 make_microcode_list last_addr in_file
;;

let (addrs,patts) = load_microcode_list 0;;

%---------------------------------------------------------------------------
|
|   let mk_ROM = new_list_rec_definition
|     (`mk_ROM`,
|      "(      mk_ROM [] (a:word9) (p:word27) = T          ) /\
|       (!aps. mk_ROM (CONS ap aps) a p =
|              (FST ap = a) => (p = SND ap) | mk_ROM aps a p)
|      ");;
|
|   letrec term_zip as ps =
|       (null as & null ps) => "[]:(word9#word27) list" |
|       "CONS (^(hd as),^(hd ps)) ^(term_zip (tl as) (tl ps))";;
|
|   let ROM = new_definition (`ROM`, "ROM a p = mk_ROM ^aps a p")
|   where aps = term_zip addrs patts;;
|
|   The theorems given below are derivable from these definitions.
|   To save work the following theorems are not derived but are
|   declared using mk_thm.
---------------------------------------------------------------------------%
```

```
new_constant (`ROM_fun`,":word9 -> word27");;

letrec mk_ROM_fun_thms as vs =
  ( (null as & null vs) => [] |
    mk_thm([], "ROM_fun ^(hd as) = ^(hd vs)") . mk_ROM_fun_thms (tl as) (tl vs)
  ) ? failwith `mk_ROM_fun_thms`;;

let ROM_fun_thms = mk_ROM_fun_thms addrs patts;;

let ROM_fun_thm = save_thm(`ROM_fun_thm`, LIST_CONJ ROM_fun_thms);;

letrec save_thm_list name n thms =
  (null thms) => ()
               | ( save_thm(name^(tok_of_int n), hd thms)
                 ; save_thm_list name (n+1) (tl thms)
                 );;

save_thm_list `ROM_fun` 0 ROM_fun_thms;;

save_thm (`ROM_fun399`,mk_thm([], "ROM_fun #110001111 = #00000000000000000000000000"));;

close_theory ();;

print_theory `-`;;
```

3.1.2 rt_CU

```
%                                                                      %
% FILE: rt_CU.ml                                                       %
%                                                                      %
% DESCRIPTION: This is the register-transfer definition of the        %
%              control unit.                                           %
%                                                                      %
% USES FILES:  microcode.th                                           %
%                                                                      %
% Brian Graham 88.04.21                                               %
%                                                                      %
% 88.04.22 - CU_DECODE loading correctly                              %
% Modifications:                                                      %
% 88.06.23 - Changed the idle and error signals to 2 flag signals     %
% 88.08.08 - Added wmem_bar signal to CU.                             %
% 88.11.07 - Moved !t outside the let expressions in DECODE.          %
%            Changed Clocked parameter from "(SUC t)" to "t".         %
% 89.07.17 - redefined using new "wordn" type                         %
%          - eliminated shift registers from this view                %
%          - brought stack contents out as parameters,                %
%            state now consists of a 5-tuple                          %
%          - redefined latches to include reset, load signals         %
% 89.09.01 - Altered clock phase labelling in the documentation.      %
%          -  Changed definition of ROM_t for unwinding.              %
%                                                                      %
%======================================================================%

new_theory `rt_CU`;;

loadf `load_wordn`;;

map new_parent [`microcode`];;

let ftime = ":num"
and mtime = ":num"
and ctime = ":num";;

let fsig = ":^ftime->bool"
and msig = ":^mtime->bool"
and csig = ":^ctime->bool";;

let w9_fvec = ":^ftime->word9"
and w9_mvec = ":^mtime->word9"
and w9_cvec = ":^ctime->word9";;

let w27_fvec = ":^ftime->word27"
and w27_mvec = ":^mtime->word27"
and w27_cvec = ":^ctime->word27";;

let fvec = ":num->^fsig"
and mvec = ":num->^msig"
and cvec = ":num->^csig";;
```

```
%-------------------------------------------------------------%
%               CONTROL UNIT - MPC latch                      %
%-------------------------------------------------------------%
%
let latch9 = new_definition
  ('latch9',
   "! (Clocked:^mtime->bool) (in_sig:^w9_mvec) (out_sig:^w9_mvec).
    latch9 Clocked in_sig out_sig =
      !t. out_sig (SUC t) = Clocked t => in_sig t | out_sig t"
  );;
%

let MPC9 = new_definition
  ('MPC9',
   "! (Clocked:^msig) (reset:^msig) (in_sig:^w9_mvec) (out_sig:^w9_mvec).
    MPC9 Clocked reset in_sig out_sig =
    !t:^mtime. out_sig (SUC t) = (Clocked t) =>
                                  (reset t) => #000000000 | in_sig t
                                             | out_sig t"
  );;

let S_latch9 = new_definition
  ('S_latch9',
   "! (Clocked:^msig) (load:^msig) (in_sig:^w9_mvec) (out_sig:^w9_mvec).
    S_latch9 Clocked load in_sig out_sig =
     !t:^mtime. out_sig (SUC t) = Clocked t =>
                                    load t => in_sig t | out_sig t
                                            | out_sig t"
  );;


let STATE_REG = new_definition
  ('STATE_REG',
   "!next_mpc next_s0 next_s1 next_s2 next_s3 mpc s0 s1 s2 s3:^w9_mvec
    (Clocked:^msig) (reset:^msig) (load:^msig) .
    STATE_REG Clocked reset load
         (next_mpc,next_s0,next_s1,next_s2,next_s3)
         (     mpc,     s0,     s1,     s2,     s3) =
     (MPC9     Clocked reset next_mpc mpc) /\
     (S_latch9 Clocked load  next_s0  s0)  /\
     (S_latch9 Clocked load  next_s1  s1)  /\
     (S_latch9 Clocked load  next_s2  s2)  /\
     (S_latch9 Clocked load  next_s3  s3)     "
  );;

%-------------------------------------------------------------%
%                  CONTROL UNIT - ROM                         %
%-------------------------------------------------------------%
% ========== this is defined in microcode ... ============== %
```

```
%-------------------------------------------------------------%
%               CONTROL UNIT - DECODE UNIT                    %
%-------------------------------------------------------------%

let CU_DECODE = new_definition
  (`CU_DECODE`,
   "CU_DECODE
                                        % inputs %
        (button:^msig)

        (mpc:^w9_mvec)
        (s0:^w9_mvec)(s1:^w9_mvec)(s2:^w9_mvec)(s3:^w9_mvec)

        (rom_out:^w27_mvec) (opcode:^w9_mvec)

        (atomflag:^msig)    (bit30flag:^msig)   (bit31flag:^msig)
        (zeroflag:^msig)    (nilflag:^msig)     (eqflag:^msig)
        (leqflag:^msig)
                                        % outputs %
        (flag0:^msig) (flag1:^msig)
        (nextmpc:^w9_mvec)
        (next_s0:^w9_mvec)  (next_s1:^w9_mvec)
        (next_s2:^w9_mvec)  (next_s3:^w9_mvec)

        (push_or_pop:^msig)

        (ralu:^msig)   (rmem:^msig)   (rarg:^msig)
        (rbuf1:^msig)  (rbuf2:^msig)  (rcar:^msig)
        (rs:^msig)     (re:^msig)     (rc:^msig)
        (rd:^msig)     (rmar:^msig)   (rx1:^msig)
        (rx2:^msig)    (rfree:^msig)  (rparent:^msig)
        (rroot:^msig)  (ry1:^msig)    (ry2:^msig)
        (rnum:^msig)   (rnil:^msig)   (rtrue:^msig)
        (rfalse:^msig) (rcons:^msig)

        (write_bit:^msig)              (wmem_bar:^msig)
        (warg:^msig)   (wbuf1:^msig)
        (wbuf2:^msig)  (wcar:^msig)    (ws:^msig)
        (we:^msig)     (wc:^msig)      (wd:^msig)
        (wmar:^msig)   (wx1:^msig)     (wx2:^msig)
        (wfree:^msig)  (wparent:^msig) (wroot:^msig)
        (wy1:^msig)    (wy2:^msig)

        (dec:^msig)      (add:^msig)      (sub:^msig)
        (mul:^msig)      (div:^msig)      (rem:^msig)
        (setbit30:^msig) (setbit31:^msig) (resetbit31:^msig)
        (replcar:^msig)  (replcdr:^msig)  (resetbit30:^msig)
        =
 !t:^mtime.
  let mpc_plus_1   = Inc9 (mpc t) in
  let write_bits   = Write_field (rom_out t) in
  let read_bits    = Read_field  (rom_out t) in
  let alu_bits     = Alu_field   (rom_out t) in
  let test         = Test_field  (rom_out t) in
```

```
let A_address    = A_field     (rom_out t) in
let idle_state   = mpc t = #000010110   in          % 22 %
let error_state  = mpc t = #000011000   in          % 24 %
let toc_state    = mpc t = #000101011   in          % 43 %
let selA         = ( (test = #0001)                \/
                     ((test = #0011) /\ bit30flag t) \/
                     ((test = #0100) /\ bit31flag t) \/
                     ((test = #0101) /\ eqflag    t) \/
                     ((test = #0110) /\ leqflag   t) \/
                     ((test = #0111) /\ nilflag   t) \/
                     ((test = #1000) /\ atomflag  t) \/
                     ((test = #1001) /\ zeroflag  t) \/
                     ((test = #1010) /\ button    t) \/
                      (test = #1011))    in
let pop           = (test = #1100)      in
let push          = (test = #1011)      in
let selOp         = (test = #0010)      in
let selNxt        = ~(selA \/ pop \/ push \/ selOp) in

(   (flag0 t      = idle_state \/ error_state)                /\
    (flag1 t      = toc_state  \/ error_state)                /\
    (nextmpc t    = (selA)   =>   A_address |
                    ((pop)    =>   s0 t    |
                     ((selOp)  =>   opcode t    |
                    %(selNxt t) %    mpc_plus_1)))            /\
    ((next_s0 t, next_s1 t, next_s2 t, next_s3 t) =
                      push => (mpc_plus_1, s0 t, s1 t, s2 t) |
                      pop  => (s1 t, s2 t, s3 t, #000000000) |
                              (s0 t, s1 t, s2 t, s3 t))       /\
   (push_or_pop t = push \/ pop)                              /\
   (ralu       t = (read_bits = #00001))                      /\
   (rmem       t = (read_bits = #00010))                      /\
   (rarg       t = (read_bits = #00011))                      /\
   (rbuf1      t = (read_bits = #00100))                      /\
   (rbuf2      t = (read_bits = #00101))                      /\
   (rcar       t = (read_bits = #00110))                      /\
   (rs         t = (read_bits = #00111))                      /\
   (re         t = (read_bits = #01000))                      /\
   (rc         t = (read_bits = #01001))                      /\
   (rd         t = (read_bits = #01010))                      /\
   (rmar       t = (read_bits = #01011))                      /\
   (rx1        t = (read_bits = #01100))                      /\
   (rx2        t = (read_bits = #01101))                      /\
   (rfree      t = (read_bits = #01110))                      /\
   (rparent    t = (read_bits = #01111))                      /\
   (rroot      t = (read_bits = #10000))                      /\
   (ry1        t = (read_bits = #10001))                      /\
   (ry2        t = (read_bits = #10010))                      /\
   (rnum       t = (read_bits = #10011))                      /\
   (rnil       t = (read_bits = #10100))                      /\
   (rtrue      t = (read_bits = #10101))                      /\
   (rfalse     t = (read_bits = #10110))                      /\
   (rcons      t = (read_bits = #10111))                      /\
```

```
     (write_bit  t = ~(write_bits = #00001))                  /\ % nand wmem with PhiA %
     (wmem_bar   t = ~(write_bits = #00001))                  /\ % nand wmem with ~PhiB %
     (warg       t = (write_bits = #00010))                   /\
     (wbuf1      t = (write_bits = #00011))                   /\
     (wbuf2      t = (write_bits = #00100))                   /\
     (wcar       t = (write_bits = #00101))                   /\
     (ws         t = (write_bits = #00110))                   /\
     (we         t = (write_bits = #00111))                   /\
     (wc         t = (write_bits = #01000))                   /\
     (wd         t = (write_bits = #01001))                   /\
     (wmar       t = (write_bits = #01010))                   /\
     (wx1        t = (write_bits = #01011))                   /\
     (wx2        t = (write_bits = #01100))                   /\
     (wfree      t = (write_bits = #01101))                   /\
     (wparent    t = (write_bits = #01110))                   /\
     (wroot      t = (write_bits = #01111))                   /\
     (wy1        t = (write_bits = #10000))                   /\
     (wy2        t = (write_bits = #10001))                   /\

     (dec        t = (alu_bits = #0001))                      /\
     (add        t = (alu_bits = #0010))                      /\
     (sub        t = (alu_bits = #0011))                      /\
     (mul        t = (alu_bits = #0100))                      /\
     (div        t = (alu_bits = #0101))                      /\
     (rem        t = (alu_bits = #0110))                      /\
     (setbit30   t = (alu_bits = #0111))                      /\
     (setbit31   t = (alu_bits = #1000))                      /\
     (resetbit31 t = (alu_bits = #1001))                      /\
     (replcar    t = (alu_bits = #1010))                      /\
     (replcdr    t = (alu_bits = #1011))                      /\
     (resetbit30 t = (alu_bits = #1100)))
");;

%-------------------------------------------------------------%
% The ROM requires time varying input and output.             %
%-------------------------------------------------------------%
let ROM_t = new_definition
  (`ROM_t`,
   "ROM_t in out = !t:^mtime.  (out t) = ROM_fun (in t)"
  );;
```

```
%-------------------------------------------------------------%
%                TOP LEVEL CONTROL UNIT                       %
%-------------------------------------------------------------%

let CU = new_definition
  ('CU',
   "!    (Clocked:^mtime->bool)
         (reset:^msig) (button:^msig)

         (mpc:^w9_mvec)
         (s0:^w9_mvec)(s1:^w9_mvec)(s2:^w9_mvec)(s3:^w9_mvec)

         (opcode:^w9_mvec)

         (atomflag:^msig) (bit30flag:^msig) (bit31flag:^msig)
         (zeroflag:^msig) (nilflag:^msig)   (eqflag:^msig)
         (leqflag:^msig)
                                          % outputs %
         (flag0:^msig) (flag1:^msig)

         (ralu:^msig)   (rmem:^msig)   (rarg:^msig)
         (rbuf1:^msig)  (rbuf2:^msig)  (rcar:^msig)
         (rs:^msig)     (re:^msig)     (rc:^msig)
         (rd:^msig)     (rmar:^msig)   (rx1:^msig)
         (rx2:^msig)    (rfree:^msig)  (rparent:^msig)
         (rroot:^msig)  (ry1:^msig)    (ry2:^msig)
         (rnum:^msig)   (rnil:^msig)   (rtrue:^msig)
         (rfalse:^msig) (rcons:^msig)

         (write_bit:^msig)              (bidir:^msig)
         (warg:^msig)   (wbuf1:^msig)
         (wbuf2:^msig)  (wcar:^msig)    (ws:^msig)
         (we:^msig)     (wc:^msig)      (wd:^msig)
         (wmar:^msig)   (wx1:^msig)     (wx2:^msig)
         (wfree:^msig)  (wparent:^msig) (wroot:^msig)
         (wy1:^msig)    (wy2:^msig)

         (dec:^msig)      (add:^msig)      (sub:^msig)
         (mul:^msig)      (div:^msig)      (rem:^msig)
         (setbit30:^msig) (setbit31:^msig) (resetbit31:^msig)
         (replcar:^msig)  (replcdr:^msig)  (resetbit30:^msig)
         .
    CU   Clocked
         reset button
         mpc s0 s1 s2 s3
         opcode
         atomflag bit30flag bit31flag zeroflag nilflag eqflag leqflag
         flag0 flag1
         ralu rmem rarg rbuf1 rbuf2 rcar rs re rc rd rmar rx1 rx2
         rfree rparent rroot ry1 ry2 rnum rnil rtrue rfalse rcons
         write_bit bidir
         warg wbuf1 wbuf2 wcar ws we wc wd wmar wx1 wx2
         wfree wparent wroot wy1 wy2
         dec add sub mul div rem
```

```
        setbit30 setbit31 resetbit31 replcar replcdr resetbit30
        =
     ? (rom_out:^w27_mvec)
       (nextmpc:^w9_mvec)
       (next_s0:^w9_mvec)
       (next_s1:^w9_mvec)
       (next_s2:^w9_mvec)
       (next_s3:^w9_mvec)
       (push_or_pop:^msig)
     .

      (STATE_REG Clocked reset push_or_pop
             (nextmpc, next_s0, next_s1, next_s2, next_s3)
             (    mpc,      s0,      s1,      s2,      s3)) /\

      (ROM_t mpc rom_out) /\

      (CU_DECODE
         button
         mpc s0 s1 s2 s3
         rom_out opcode
         atomflag bit30flag bit31flag zeroflag nilflag eqflag leqflag
                                        % outputs %
         flag0 flag1
         nextmpc next_s0 next_s1 next_s2 next_s3
         push_or_pop
         ralu rmem rarg rbuf1 rbuf2 rcar rs re rc rd rmar rx1 rx2
         rfree rparent rroot ry1 ry2 rnum rnil rtrue rfalse rcons
         write_bit bidir
         warg wbuf1 wbuf2 wcar ws we wc wd wmar wx1 wx2
         wfree wparent wroot wy1 wy2
         dec add sub mul div rem
         setbit30 setbit31 resetbit31 replcar replcdr resetbit30
    )"
  );;

close_theory ();;

print_theory `-`;;
```

3.1.3 interface

```
% SECD verification                                                    %
%                                                                     %
% FILE:        interface.ml                                           %
%                                                                     %
% DESCRIPTION: This includes definitions used by components at        %
%              the rt level view of the host machine.                 %
%                                                                     %
% USES FILES:                                                         %
%                                                                     %
% Brian Graham 89.09.05                                               %
%                                                                     %
% Modifications:                                                      %
%=====================================================================%

new_theory `interface`;;


let ftime = ":num"
and mtime = ":num"
and ctime = ":num";;

let fsig = ":^ftime->bool"
and msig = ":^mtime->bool"
and csig = ":^ctime->bool";;

%-------------------- define one_asserted_12 -----------------%
% this is used by the ALU: only one operation at a time...   %

let one_asserted_12 = new_definition
  (`one_asserted_12`,
   "!a b c d e f g h i j k l:^msig.
    one_asserted_12 a b c d e f g h i j k l =
      !t.
          (a t ==> (~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\ ~g t
                        /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t)) /\
          (b t ==> (~a t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\ ~g t
                        /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t)) /\
          (c t ==> (~a t /\ ~b t /\ ~d t /\ ~e t /\ ~f t /\ ~g t
                        /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t)) /\
          (d t ==> (~a t /\ ~b t /\ ~c t /\ ~e t /\ ~f t /\ ~g t
                        /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t)) /\
          (e t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~f t /\ ~g t
                        /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t)) /\
          (f t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~g t
                        /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t)) /\
          (g t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t
                        /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t)) /\
          (h t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t
                        /\ ~g t /\ ~i t /\ ~j t /\ ~k t /\ ~l t)) /\
          (i t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t
                        /\ ~g t /\ ~h t /\ ~j t /\ ~k t /\ ~l t)) /\
          (j t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t
```

```
                    /\ ~g t /\ ~h t /\ ~i t /\ ~k t /\ ~l t)) /\
        (k t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t
                    /\ ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~l t)) /\
        (l t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t
                    /\ ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~k t))"
  );;

let one_asserted_23 = new_definition
  (`one_asserted_23`,
   "!a b c d e f g h i j k l m n p q r s u v w x y :^msig.
    one_asserted_23 a b c d e f g h i j k l m n p q r s u v w x y =
      !t.
          (a t ==> (~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\ ~g t /\
                    ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\ ~m t /\
                    ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                    ~v t /\ ~w t /\ ~x t /\ ~y t))                 /\
          (b t ==> (~a t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\ ~g t /\
                    ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\ ~m t /\
                    ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                    ~v t /\ ~w t /\ ~x t /\ ~y t))                 /\
          (c t ==> (~a t /\ ~b t /\ ~d t /\ ~e t /\ ~f t /\ ~g t /\
                    ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\ ~m t /\
                    ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                    ~v t /\ ~w t /\ ~x t /\ ~y t))                 /\
          (d t ==> (~a t /\ ~b t /\ ~c t /\ ~e t /\ ~f t /\ ~g t /\
                    ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\ ~m t /\
                    ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                    ~v t /\ ~w t /\ ~x t /\ ~y t))                 /\
          (e t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~f t /\ ~g t /\
                    ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\ ~m t /\
                    ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                    ~v t /\ ~w t /\ ~x t /\ ~y t))                 /\
          (f t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~g t /\
                    ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\ ~m t /\
                    ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                    ~v t /\ ~w t /\ ~x t /\ ~y t))                 /\
          (g t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                    ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\ ~m t /\
                    ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                    ~v t /\ ~w t /\ ~x t /\ ~y t))                 /\
          (h t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                    ~g t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\ ~m t /\
                    ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                    ~v t /\ ~w t /\ ~x t /\ ~y t))                 /\
          (i t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                    ~g t /\ ~h t /\ ~j t /\ ~k t /\ ~l t /\ ~m t /\
                    ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                    ~v t /\ ~w t /\ ~x t /\ ~y t))                 /\
          (j t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                    ~g t /\ ~h t /\ ~i t /\ ~k t /\ ~l t /\ ~m t /\
                    ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                    ~v t /\ ~w t /\ ~x t /\ ~y t))                 /\
          (k t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                    ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~l t /\ ~m t /\
```

```
                ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                ~v t /\ ~w t /\ ~x t /\ ~y t))                  /\
          (l t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~m t /\
                ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                ~v t /\ ~w t /\ ~x t /\ ~y t))                  /\
          (m t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\
                ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                ~v t /\ ~w t /\ ~x t /\ ~y t))                  /\
          (n t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\
                ~m t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                ~v t /\ ~w t /\ ~x t /\ ~y t))                  /\
          (p t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\
                ~m t /\ ~n t /\ ~q t /\ ~r t /\ ~s t /\ ~u t /\
                ~v t /\ ~w t /\ ~x t /\ ~y t))                  /\
          (q t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\
                ~m t /\ ~n t /\ ~p t /\ ~r t /\ ~s t /\ ~u t /\
                ~v t /\ ~w t /\ ~x t /\ ~y t))                  /\
          (r t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\
                ~m t /\ ~n t /\ ~p t /\ ~q t /\ ~s t /\ ~u t /\
                ~v t /\ ~w t /\ ~x t /\ ~y t))                  /\
          (s t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\
                ~m t /\ ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~u t /\
                ~v t /\ ~w t /\ ~x t /\ ~y t))                  /\
          (u t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\
                ~m t /\ ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\
                ~v t /\ ~w t /\ ~x t /\ ~y t))                  /\
          (v t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\
                ~m t /\ ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\
                ~u t /\ ~w t /\ ~x t /\ ~y t))                  /\
          (w t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\
                ~m t /\ ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\
                ~u t /\ ~v t /\ ~x t /\ ~y t))                  /\
          (x t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\
                ~m t /\ ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\
                ~u t /\ ~v t /\ ~w t /\ ~y t))                  /\
          (y t ==> (~a t /\ ~b t /\ ~c t /\ ~d t /\ ~e t /\ ~f t /\
                ~g t /\ ~h t /\ ~i t /\ ~j t /\ ~k t /\ ~l t /\
                ~m t /\ ~n t /\ ~p t /\ ~q t /\ ~r t /\ ~s t /\
                ~u t /\ ~v t /\ ~w t /\ ~x t))
"
 );;

close_theory ();;
```

```
print_theory `-`;;

%=====================================================================%
% These definitions are possible alternatives to the above           %
% very specialized ones.                                             %
%=====================================================================%

%
let one_asserted_fun = new_list_rec_definition
(`one_asserted_fun`,
 "(one_asserted_fun [] f = T) /\
  (one_asserted_fun (CONS h t) f =
          f => (~h /\ (one_asserted_fun t f))
            |  (one_asserted_fun t h))");;

let one_asserted = new_definition
(`one_asserted`, "one_asserted l = one_asserted_fun l F");;
%
```

3.1.4 rt_DP

```
% SECD verification                                                    %
%                                                                     %
% FILE:        rt_DP.ml                                               %
%                                                                     %
% DESCRIPTION: This is a first attempt to define an intermediate      %
%              level view of the host machine, aimed at a             %
%              register transfer level of the data path.              %
%                                                                     %
% USES FILES:  dp_types.th                                            %
%                                                                     %
% Brian Graham 88.02.03                                               %
%                                                                     %
% Modifications:                                                      %
% 88.03.02 - added busgates between pads and bus                      %
% 88.05.11 - added Clocked predicate to registers                     %
% 88.10.07 - made x1,x2,buf1 not hidden.                              %
% 89.09.05 - removed one_asserted_12 definition.                      %
% 89.09.07 - changed recognizer fun's from ":word2->bool" to          %
%            ":word32->bool".                                         %
%          - moved recognizer and field selector functions to         %
%            dp_types to allow access from mem_abs.ml.                %
%          - redefined eq_flag as implemented.                        %
%          - defined all ALU operations on 32 bit words, in terms     %
%            of actual function, or undefined integer operations.     %
% ================================================================== %
new_theory `rt_DP`;;

loadf  `load_wordn`;;

map new_parent [`dp_types`; `interface`; `modulo_ops`];;

load_definitions `dp_types`;;

map (load_theorem `dp_types`)
    [`Bits14_Word14`; `Bits2_Word2`];;
load_definition `interface` `one_asserted_12`;;

let ftime = ":num"
and mtime = ":num"
and ctime = ":num";;

let fsig = ":^ftime->bool"
and msig = ":^mtime->bool"
and csig = ":^ctime->bool";;

let w2_fvec = ":^ftime->word2"
and w2_mvec = ":^mtime->word2"
and w2_cvec = ":^ctime->word2";;

let w14_fvec = ":^ftime->word14"
and w14_mvec = ":^mtime->word14"
and w14_cvec = ":^ctime->word14";;
```

```
let w28_fvec = ":^ftime->word28"
and w28_mvec = ":^mtime->word28"
and w28_cvec = ":^ctime->word28";;

let w32_fvec = ":^ftime->word32"
and w32_mvec = ":^mtime->word32"
and w32_cvec = ":^ctime->word32";;

% ================= constructor function ======================== %
% Note the default always sets the gc bits to #00.                 %
% ================================================================ %
let mk_num_record = new_definition
  (`mk_num_record`,
   "!n:word28.
    mk_num_record n =
        Word32 (Bus F (Bus F (Concat (Bits2  RT_NUMBER)
                                     (Bits28 n)        )))"
  );;

let bus32_cons_append = new_definition
  (`bus32_cons_append`,
   "! (a:word2) (b:word2)(c:word14)(d:word14).
    bus32_cons_append a b c d =
        Word32 (Concat (Bits2 a)
                       (Concat (Bits2 b)
                               (Concat (Bits14 c) (Bits14 d))))"
  );;

let bus32_num_append = new_definition
  (`bus32_num_append`,
   "! (a:word28).
    bus32_num_append a =
        Word32 (Concat (Bits2 #00)
                       (Concat (Bits2 RT_NUMBER)
                               (Bits28 a)))"
  );;

let gc_bus32_append = new_definition
  (`gc_bus32_append`,
   "! (a:bool) (b:bool)(c:word32).
    gc_bus32_append a b c =
        Word32 (Bus a (Bus b (Tl_bus (Tl_bus (Bits32 c)))))"
  );;
```

```
% ================= useful lemma ??????? ======================== %
let bus32_cons_append_lemma = prove_thm
  (`bus32_cons_append_lemma`,
   "bus32_cons_append (Word2 (Bus b32 (Wire b31)))
                      (Word2 (Bus b30 (Wire b29)))
                      (Word14 (Bus b28 (Bus b27 (Bus b26 (Bus b25
                              (Bus b24 (Bus b23 (Bus b22 (Bus b21
                              (Bus b20 (Bus b19 (Bus b18 (Bus b17
                              (Bus b16 (Wire b15))))))))))))))
                      (Word14 (Bus b14 (Bus b13 (Bus b12 (Bus b11
                              (Bus b10 (Bus b9  (Bus b8  (Bus b7
                              (Bus b6  (Bus b5  (Bus b4  (Bus b3
                              (Bus b2  (Wire (b1:bool))))))))))))))) =
        Word32    (Bus b32 (Bus b31 (Bus b30 (Bus b29 (Bus b28
                  (Bus b27 (Bus b26 (Bus b25 (Bus b24 (Bus b23
                  (Bus b22 (Bus b21 (Bus b20 (Bus b19 (Bus b18
                  (Bus b17 (Bus b16 (Bus b15 (Bus b14 (Bus b13
                  (Bus b12 (Bus b11 (Bus b10 (Bus b9  (Bus b8
                  (Bus b7  (Bus b6  (Bus b5  (Bus b4  (Bus b3
                  (Bus b2  (Wire (b1:bool) ))))))))))))))))))))))))))))))
    ",
   prt [Bits14_Word14; Bits2_Word2; bus32_cons_append;
        definition `bus` `Concat`]
   THEN REFL_TAC
   );;

%-------------------------------------------------------------%
%                   primitive gates                           %
%-------------------------------------------------------------%

let busgate2 = new_definition
  (`busgate2`,
   "! (in_val:^w2_mvec) (rd:^msig) (out:^w2_mvec).
    busgate2 in_val rd out =
      !t. (rd t) ==> (out t = in_val t)"
  );;

let busgate14 = new_definition
  (`busgate14`,
   "! (in_val:^w14_mvec) (rd:^msig) (out:^w14_mvec).
    busgate14 in_val rd out =
      !t. (rd t) ==> (out t = in_val t)"
  );;

let busgate32 = new_definition
  (`busgate32`,
   "! (in_val:^w32_mvec) (rd:^msig) (out:^w32_mvec).
    busgate32 in_val rd out =
      !t. (rd t) ==> (out t = in_val t)"
  );;
```

```
%-------------------------------------------------------------%
let reg2 = new_definition
  (`reg2`,
   "! (in_sig:^w2_mvec) (out_sig:^w2_mvec) (clocked:^mtime->bool)
      (wr:^msig)        (rd:^msig)         (st:^w2_mvec) .
    reg2 in_sig out_sig clocked wr rd st =
      (!t:^mtime.
         st(t+1) = clocked(t+1) => (wr(t+1)=>in_sig(t+1)|(st t))
                                 | (st t)
      ) /\
      (!t:^mtime. (rd t) ==> ((out_sig t) = (st t)))"
  );;

%-----------------------------------------------------------------------
|   register is a reg with both input and output joinde to same node.
-----------------------------------------------------------------------%
let register2 = new_definition
  (`register2`,
   "! (sgl:^w2_mvec).  register2 sgl = reg2 sgl sgl"
  );;

let reg14 = new_definition
  (`reg14`,
   "! (in_sig:^w14_mvec) (out_sig:^w14_mvec) (clocked:^mtime->bool)
      (wr:^msig)         (rd:^msig)          (st:^w14_mvec) .
    reg14 in_sig out_sig clocked wr rd st =
      (!t:^mtime.
         st(t+1) = clocked (t+1) => (wr(t+1)=>in_sig(t+1)|(st t))
                                  | (st t)
      ) /\
      (!t:^mtime. (rd t) ==> ((out_sig t) = (st t)))"
  );;

let register14 = new_definition
  (`register14`,
   "! (sgl:^w14_mvec).  register14 sgl = reg14 sgl sgl"
  );;

let reg32 = new_definition
  (`reg32`,
   "! (in_sig:^w32_mvec) (out_sig:^w32_mvec) (clocked:^mtime->bool)
      (wr:^msig) (rd:^msig) (st:^w32_mvec).
    reg32 in_sig out_sig clocked wr rd st =
      (!t:^mtime.
         st(t+1) = clocked(t+1) => (wr(t+1)=>in_sig(t+1)|(st t))
                                 | (st t)
      ) /\
      (!t:^mtime. (rd t) ==> ((out_sig t) = (st t)))"
  );;

let register32 = new_definition
  (`register32`,
   "! (sgl:^w32_mvec).  register32 sgl = reg32 sgl sgl"
  );;
```

```
%-------------------------------------------------------------%
%                          busgates                           %
%-------------------------------------------------------------%

let NUM = new_definition
  (`NUM`,
   "! (rnum:^msig) (a_sig:^w14_mvec) (d_sig:^w14_mvec).
    NUM rnum a_sig d_sig =
        (busgate14 (\t:num. ZEROS14) rnum a_sig) /\
        (busgate14 (\t:num. NUM_addr)  rnum d_sig)"
  );;

let Nil = new_definition
  (`Nil`,
   "Nil = busgate14 (\t:^mtime. NIL_addr)"
  );;

let TRUE = new_definition
  (`TRUE`,
   "TRUE = busgate14 (\t:^mtime. T_addr)"
  );;

let FALSE = new_definition
  (`FALSE`,
   "FALSE = busgate14 (\t:^mtime. F_addr)"
  );;

%-------------------------------------------------------------%
%                    state registers                          %
%-------------------------------------------------------------%
let s = new_definition
  (`s`, "s = register14" );;

let e = new_definition
  (`e`, "e = register14" );;

let c = new_definition
  (`c`, "c = register14" );;

let d = new_definition
  (`d`, "d = register14" );;
```

```
%-------------------------------------------------------------%
%                  misc. registers                            %
%-------------------------------------------------------------%
let READ_MEM = new_definition
  (`READ_MEM`, "READ_MEM = busgate32" );;

let MAR = new_definition
  (`MAR`,
   "!(c_sig:^w14_mvec) (d_sig:^w14_mvec) (clocked:^mtime->bool)
     (wmar:^msig)      (rmar:^msig)      (mar:^w14_mvec) .
    MAR c_sig d_sig clocked wmar rmar mar =
        (register14 d_sig clocked wmar rmar mar)   /\
        (busgate14 (\t:^mtime. ZEROS14) rmar c_sig)");;

let FREE = new_definition
  (`FREE`, "FREE = register14" );;

let PARENT = new_definition
  (`PARENT`, "PARENT = register14" );;

let ROOT = new_definition
  (`ROOT`, "ROOT = register14" );;

let Y1 = new_definition
  (`Y1`, "Y1 = register14" );;

let X1 = new_definition
  (`X1`, "X1 = register14");;

let X2 = new_definition
  (`X2`, "X2 = register14" );;

let Car = new_definition
  (`Car`,
   "! (c_sig:^w14_mvec) (d_sig:^w14_mvec).
    Car c_sig d_sig = reg14 c_sig d_sig");;

let Y2 = new_definition
  (`Y2`, "Y2 = register14" );;

let ARG = new_definition
  (`ARG`, "ARG = register32" );;

let BUF1 = new_definition
  (`BUF1`,
   "! (in_sig:^w32_mvec) (out_sig:^w32_mvec).
    BUF1 in_sig out_sig = reg32 in_sig out_sig ");;

let BUF2 = new_definition
  (`BUF2`,
   "! (in_sig:^w32_mvec) (out_sig:^w32_mvec).
    BUF2 in_sig out_sig = reg32 in_sig out_sig ");;
```

```
%-------------------------------------------------------------%
%                          cons unit                          %
%-------------------------------------------------------------%
let Cons = new_definition
  ('Cons',
   "! (c_in_sig:^w14_mvec) (d_in_sig:^w14_mvec)
      (rcons:^msig)
      (bus:^w32_mvec).
    Cons c_in_sig d_in_sig rcons bus =
      let g_out_sig = (\t:^mtime. garbage_bits  (bus t))
      and r_out_sig = (\t:^mtime. rec_type_bits (bus t))
      and c_out_sig = (\t:^mtime. car_bits      (bus t))
      and d_out_sig = (\t:^mtime. cdr_bits      (bus t))
      in
      ((busgate2 (\t:^mtime. #00)     rcons g_out_sig) /\
       (busgate2 (\t:^mtime. RT_CONS) rcons r_out_sig) /\
       (busgate14 c_in_sig            rcons c_out_sig) /\
       (busgate14 d_in_sig            rcons d_out_sig))"
  );;

%-------------------------------------------------------------%
%                        flagsunit                            %
%-------------------------------------------------------------%
let LEQ_prim = new_definition
('LEQ_prim',
 "LEQ_prim (x:word32) (y:word32) =
    let ival_x = (iVal o Bits28 o atom_bits) x
    in
    let ival_y = (iVal o Bits28 o atom_bits) y
    in
    ((ival_x below ival_y) \/ (ival_x = ival_y))"
 );;

let FLAGSUNIT = new_definition
  ('FLAGSUNIT',
   "! (bus:^w32_mvec)  (arg:^w32_mvec)
      (atomflag:^msig) (bit30flag:^msig) (bit31flag:^msig)
      (zeroflag:^msig) (nilflag:^msig)   (eqflag:^msig)
      (leqflag:^msig).
    FLAGSUNIT bus arg
      atomflag bit30flag bit31flag zeroflag nilflag eqflag leqflag =
    !t.
      (atomflag  t = is_atom (bus t))                  /\
      (bit30flag t = field_bit (bus t))                /\
      (bit31flag t = mark_bit (bus t))                 /\
      (zeroflag  t = (atom_bits (bus t) = ZERO28) )    /\
      (nilflag   t = (cdr_bits (bus t) = NIL_addr))    /\
      (eqflag    t = (bus t = arg t))                  /\
      (leqflag t   = LEQ_prim (arg t) (bus t))"     % arg <= bus %
  );;
```

```
%-------------------------------------------------------------%
%                          alu                                %
%-------------------------------------------------------------%
let REPLCAR = new_definition
('REPLCAR',
 "REPLCAR (x:word32) (y:word14) =
    bus32_cons_append (garbage_bits x) (rec_type_bits x)
                      y                (cdr_bits x)"
 );;

let REPLCDR = new_definition
('REPLCDR',
 "REPLCDR (x:word32) (y:word14) =
    bus32_cons_append (garbage_bits x) (rec_type_bits x)
                      (car_bits x)     y"
 );;

let SUB28 = new_definition
('SUB28',
 "SUB28 (x:word28) (y:word28) =
    let ival_x = (iVal o Bits28) x
    in
    let ival_y = (iVal o Bits28) y
    in
    bus32_num_append
      (@bit28_val:word28.
         (iVal o Bits28) bit28_val =
         (ival_x modulo_28_Sub ival_y))"
 );;

let ADD28 = new_definition
('ADD28',
 "ADD28 (x:word28) (y:word28) =
    let ival_x = (iVal o Bits28) x
    in
    let ival_y = (iVal o Bits28) y
    in
    bus32_num_append
      (@bit28_val:word28.
         (iVal o Bits28) bit28_val =
         (ival_x modulo_28_Add ival_y))"
 );;

let DEC28 = new_definition
('DEC28',
 "DEC28 (x:word28) =
    let ival_x = (iVal o Bits28) x
    in
    bus32_num_append
      (@bit28_val:word28.
         (iVal o Bits28) bit28_val =
         (modulo_28_Dec ival_x))"
 );;
```

```
let SETBIT30 = new_definition
(`SETBIT30`,
 "SETBIT30 (x:word32) =
    gc_bus32_append (mark_bit x) T x"
 );;

let SETBIT31 = new_definition
(`SETBIT31`,
 "SETBIT31 (x:word32) =
    gc_bus32_append T (field_bit x) x"
 );;

let RESETBIT30 = new_definition
(`RESETBIT30`,
 "RESETBIT30 (x:word32) =
    gc_bus32_append (mark_bit x) F x"
 );;

let RESETBIT31 = new_definition
(`RESETBIT31`,
 "RESETBIT31 (x:word32) =
    gc_bus32_append F (field_bit x) x"
 );;


%<-------------------------------------------------------------------------
let DEC28 = new_definition
  (`DEC28`,
   "!x:word28. DEC28 x =
    let num_val = VAL28 x
    in
    mk_num_record ((num_val = 0) => #11111111111111111111111111111
                                  | WORD28 (PRE num_val))"
  );;
-------------------------------------------------------------------------->%
```

```
let ALU = new_definition
  ('ALU',
   "! (replcar:^msig)    (replcdr:^msig)
      (sub:^msig)        (add:^msig)        (dec:^msig)
      (mul:^msig)        (div:^msig)        (rem:^msig)
      (setbit30:^msig)   (setbit31:^msig)
      (resetbit30:^msig) (resetbit31:^msig)
      (ralu:^msig)
      (arg:^w32_mvec)    (y2:^w14_mvec)
      (bus:^w32_mvec)    (alu:^w32_mvec).
    ALU replcar replcdr sub add dec mul div rem
        setbit30 setbit31 resetbit30 resetbit31
        ralu arg y2 bus alu =
    (one_asserted_12 replcar replcdr sub add dec mul div rem
        setbit30 setbit31 resetbit30 resetbit31 ==>
      (!t.
        let bus28 = (atom_bits (bus t))
        in
        let arg28 = (atom_bits (arg t))
        in
        ((replcar t     ==> (alu t = REPLCAR (arg t) (y2 t)))    /\
         (replcdr t     ==> (alu t = REPLCDR (arg t) (y2 t)))    /\
         (sub t         ==> (alu t = SUB28    arg28   bus28))    /\
         (add t         ==> (alu t = ADD28    arg28   bus28))    /\
         (dec t         ==> (alu t = DEC28    bus28))            /\
         (mul t         ==> (alu t = DEC28    bus28))            /\
         (div t         ==> (alu t = DEC28    bus28))            /\
         (rem t         ==> (alu t = DEC28    bus28))            /\
         (setbit31 t    ==> (alu t = SETBIT31 (arg t)))          /\
         (setbit30 t    ==> (alu t = SETBIT30 (arg t)))          /\
         (resetbit31 t  ==> (alu t = RESETBIT31 (arg t)))        /\
         (resetbit30 t  ==> (alu t = RESETBIT30 (arg t))))
      )
      /\
      (!t. ralu t ==> (bus t = alu t))
    )"
  );;
```

```
%-------------------------------------------------------------%
%                  TOP LEVEL DATA PATH                        %
%-------------------------------------------------------------%
let DP = new_definition
  ('DP',
   "DP bus bus_in (Clocked:^mtime->bool)
       rmem
       mar      wmar      rmar      rnum rnil rtrue rfalse
       s'       ws        rs
       e'       we        re
       c'       wc        rc
       d'       wd        rd
       free     wfree     rfree
       parent   wparent   rparent
       root     wroot     rroot
       y1       wy1       ry1
       x1       wx1       rx1
       x2       wx2       rx2
       y2       wy2       ry2       rcons
       car      wcar      rcar
       atomflag bit30flag bit31flag zeroflag nilflag eqflag leqflag
       arg      warg      rarg
       buf1     wbuf1     rbuf1
       buf2     wbuf2     rbuf2
       replcar  replcdr   sub add dec mul div rem
       setbit30 setbit31  resetbit30 resetbit31
       ralu =
    ? (alu:^w32_mvec) .
      let a_bus = (\t. car_bits (bus t))
      and d_bus = (\t. cdr_bits (bus t))
      in
      ( (READ_MEM bus_in rmem bus)                          /\
        (MAR a_bus d_bus Clocked wmar     rmar    mar)     /\
        (NUM   rnum   a_bus d_bus)                          /\
        (Nil   rnil   d_bus)                                /\
        (TRUE  rtrue  d_bus)                                /\
        (FALSE rfalse d_bus)                                /\
        (s       d_bus   Clocked  ws      rs      s')      /\
        (e       d_bus   Clocked  we      re      e')      /\
        (c       d_bus   Clocked  wc      rc      c')      /\
        (d       d_bus   Clocked  wd      rd      d')      /\
        (FREE    d_bus   Clocked  wfree   rfree   free)    /\
        (PARENT  d_bus   Clocked  wparent rparent parent)  /\
        (ROOT    d_bus   Clocked  wroot   rroot   root)    /\
        (Y1      d_bus   Clocked  wy1     ry1     y1)      /\
        (X1      d_bus   Clocked  wx1     rx1     x1)      /\
        (X2      d_bus   Clocked  wx2     rx2     x2)      /\
        (Cons x1 x2 rcons bus)                              /\
        (Car a_bus d_bus Clocked  wcar    rcar    car)     /\
        (FLAGSUNIT bus arg atomflag bit30flag bit31flag
                   zeroflag nilflag eqflag    leqflag   ) /\
        (Y2      d_bus   Clocked  wy2     ry2     y2)      /\
        (ARG     bus     Clocked  warg    rarg    arg)     /\
        (ALU replcar replcdr sub add dec mul div rem
```

```
            setbit30 setbit31 resetbit30 resetbit31
            ralu arg y2 bus alu                          ) /\
        (BUF1    alu bus Clocked  wbuf1   rbuf1   buf1)   /\
        (BUF2    alu bus Clocked  wbuf2   rbuf2   buf2)
    )"
  );;

close_theory ();;

print_theory `-`;;
```

3.1.5 rt_PADS

```
% SECD verification                                                    %
%                                                                     %
% FILE: rt_PADS.ml                                                    %
%                                                                     %
% DESCRIPTION: This is a higher level view of the pads frame,         %
%              much more of a register transfer view of things,       %
%              mainly because of the higher level data                %
%              abstraction used.                                      %
%                                                                     %
% USES FILES:  dp_types.th                                            %
%                                                                     %
% Brian Graham 88.05.12                                               %
%                                                                     %
% Modifications:                                                      %
% 88.06.23 - Changed the idle and error signals to 2 flag signals     %
%                                                                     %
%=====================================================================%

new_theory `rt_PADS`;;

loadf `load_wordn`;;

new_parent `dp_types`;;

let ftime = ":num"
and mtime = ":num"
and ctime = ":num";;

let fsig = ":^ftime->bool"
and msig = ":^mtime->bool"
and csig = ":^ctime->bool";;

let w14_fvec = ":^ftime->word14"
and w14_mvec = ":^mtime->word14"
and w14_cvec = ":^ctime->word14";;

let w32_fvec = ":^ftime->word32"
and w32_mvec = ":^mtime->word32"
and w32_cvec = ":^ctime->word32";;
```

```
% ========================================================== %
% We treat inputs and outputs as high level signals.         %
%                                                            %
%   note that bidir is asserted LOW:                         %
%     (bidir = HI) => read from pins - read memory mode      %
%     (bidir = LO) => write to memory mode                   %
%                                                            %
% The bus connections are :                                  %
%                  -----------                               %
% bus_bits ------>|           |                              %
%                 | bidir pad |<------> bus_pins             %
% mem_bits <------|___________|                              %
%                      o                                     %
%                      |                                     %
%                    bidir                                   %
%                                                            %
% ie. mem_bits get the memory value from bus_pins on a rmem. %
% ========================================================== %
let rt_PF = new_definition
  (`rt_PF`,
   "! (reset:^msig)         (button:^msig)
      (flag0:^msig)         (flag1:^msig)
      (bidir:^msig)         (write_bit:^msig)    (rmem:^msig)
      (bus_bits:^w32_mvec)  (mem_bits:^w32_mvec) (mar_bits:^w14_mvec)

      (reset_pin:^msig)     (button_pin:^msig)
      (flag0_pin:^msig)     (flag1_pin:^msig)
      (write_bit_pin:^msig) (rmem_pin:^msig)
      (bus_pins:^w32_mvec)  (mar_pins:^w14_mvec)
    .
    rt_PF                                % chip side %
      reset         button
      flag0         flag1
      bidir         write_bit   rmem
      bus_bits      mem_bits    mar_bits
                                          % pin side %
      reset_pin     button_pin
      flag0_pin     flag1_pin
      write_bit_pin rmem_pin
      bus_pins      mar_pins
     =
    !t:^mtime.
     (reset t              = reset_pin t)  /\
     (button t             = button_pin t) /\
     (flag0_pin t          = flag0 t)      /\
     (flag1_pin t          = flag1 t)      /\
     (write_bit_pin t      = write_bit t)  /\
     (rmem_pin t           = rmem t)       /\
     (bidir t => (mem_bits t = bus_pins t)       % read from memory %
               | (bus_pins t = bus_bits t)) /\   % write to memory  %
     (mar_pins t           = mar_bits t)
   ");;

close_theory();;
```

```
load_library `unwind`;;

% ================================================================ %
% The following lemma reorders the equations and moves in the      %
% quantified "t", for use in eliminating quantified variables at   %
% a higher level in the proof.                                     %
% ================================================================ %
let rt_PF_lemma = prove_thm
(`rt_PF_lemma`,
 "rt_PF                                % chip side %
      reset         button
      flag0         flag1
      bidir         write_bit   rmem
      bus_bits      mem_bits    mar_bits
                                     % pin side %
      reset_pin     button_pin
      flag0_pin     flag1_pin
      write_bit_pin rmem_pin
      bus_pins      mar_pins
     =
     (!t:^mtime. reset t               = reset_pin t)      /\
     (!t:^mtime. button t              = button_pin t)     /\
     (!t:^mtime. flag0 t               = flag0_pin t)      /\
     (!t:^mtime. flag1 t               = flag1_pin t)      /\
     (!t:^mtime. write_bit t           = write_bit_pin t)  /\
     (!t:^mtime. rmem t                = rmem_pin t)       /\
     (!t:^mtime. bidir t => (mem_bits t = bus_pins t) |
                            (bus_bits t = bus_pins t))     /\
     (!t:^mtime. mar_bits t            = mar_pins t) ",
  port[rt_PF]
  THEN in_conv_tac UNWINDF
  THEN EQ_TAC
  THEN STRIP_TAC
  THEN art[]
  THEN GEN_TAC
  THENL
  [ SUBST1_TAC (SPECL ["(bus_bits:^w32_mvec) t"; "(bus_pins:^w32_mvec) t"]
           (INST_TYPE [":word32",":*"; ":word32",":*"] EQ_SYM_EQ ))
    ;
    SUBST1_TAC (SPECL ["(bus_pins:^w32_mvec) t"; "(bus_bits:^w32_mvec) t"]
           (INST_TYPE [":word32",":*"; ":word32",":*"] EQ_SYM_EQ ))
  ]
  THEN art[]
  );;

print_theory `-`;;
```

3.1.6 rt_SECD

```
% SECD verification                                                    %
%                                                                     %
% FILE:        rt_SECD.ml                                             %
%                                                                     %
% DESCRIPTION: This is the register transfer view of the secd         %
%              chip.  It still lacks the external memory, which       %
%              will, together with the SECD, define the SYSTEM.       %
%                                                                     %
% USES FILES:  rt_CU.th                                               %
%               rt_DP.th                                              %
%               rt_PADS.th                                            %
%                                                                     %
% Brian Graham 88.05.11                                               %
%                                                                     %
% Modifications:                                                      %
% 88.06.23 - Changed the idle and error signals to 2 flag signals     %
% 88.08.08 - Added wmem_bar signal                                    %
% 88.10.07 - x1,x2,buf1 added to DP parameters.                       %
%=====================================================================%
new_theory `rt_SECD`;;

loadf `load_wordn`;;

map new_parent [ `rt_CU`
               ; `rt_DP`
               ; `rt_PADS`
               ];;

let ftime = ":num"
and mtime = ":num"
and ctime = ":num";;

let fsig = ":^ftime->bool"
and msig = ":^mtime->bool"
and csig = ":^ctime->bool";;

let w9_fvec = ":^ftime->word9"
and w9_mvec = ":^mtime->word9"
and w9_cvec = ":^ctime->word9";;

let w14_fvec = ":^ftime->word14"
and w14_mvec = ":^mtime->word14"
and w14_cvec = ":^ctime->word14";;

let w27_fvec = ":^ftime->word27"
and w27_mvec = ":^mtime->word27"
and w27_cvec = ":^ctime->word27";;

let w32_fvec = ":^ftime->word32"
and w32_mvec = ":^mtime->word32"
and w32_cvec = ":^ctime->word32";;
```

```
let Word32 = theorem `dp_types` `Word32`;;

%-------------------------------------------------------------%
% A couple of useful selector functions.                      %
%-------------------------------------------------------------%
let w32 = "Word32 (Bus b32 (Bus b31
                  (Bus b30 (Bus b29 (Bus b28 (Bus b27 (Bus b26
                  (Bus b25 (Bus b24 (Bus b23 (Bus b22 (Bus b21
                  (Bus b20 (Bus b19 (Bus b18 (Bus b17 (Bus b16
                  (Bus b15 (Bus b14 (Bus b13 (Bus b12 (Bus b11
                  (Bus b10 (Bus b9  (Bus b8  (Bus b7  (Bus b6
                  (Bus b5  (Bus b4  (Bus b3  (Bus b2  (Wire (b1:bool)
                  ))))))))))))))))))))))))))))))))";;

let opcode_bits = new_recursive_definition false Word32
   `opcode_bits`
   "opcode_bits ^w32 =
      Word9 (Bus b9 (Bus b8 (Bus b7 (Bus b6 (Bus b5 (Bus b4
            (Bus b3 (Bus b2 (Wire (b1:bool)))))))))"
 ;;

let Opcode = new_definition
  (`Opcode`,
   "Opcode (b:^w32_mvec) (t:^mtime) = opcode_bits (b t)"
  );;

%-------------------------------------------------------------%
%                     SECD - the chip                         %
%                     ***************                         %
% This top level view of the chip has 4 functional            %
% components:  the control unit, the datapath, the shift      %
% registers (primitive testing structure) and the pad frame.  %
%-------------------------------------------------------------%
let SECD = new_definition
  (`SECD`,
   "SECD
    (SYS_Clocked:^mtime->bool)
    (mpc:^w9_mvec)
    (s0:^w9_mvec)          (s1:^w9_mvec)
    (s2:^w9_mvec)          (s3:^w9_mvec)
    (reset_pin:^msig)      (button_pin:^msig)
    (flag0_pin:^msig)      (flag1_pin:^msig)
    (write_bit_pin:^msig) (rmem_pin:^msig)
    (bus_pins:^w32_mvec)  (mar_pins:^w14_mvec)
    (s':^w14_mvec)         (e':^w14_mvec)
    (c':^w14_mvec)         (d':^w14_mvec)
    (free:^w14_mvec)
    (x1:^w14_mvec)         (x2:^w14_mvec)
    (y1:^w14_mvec)         (y2:^w14_mvec)
    (car:^w14_mvec)
    (root:^w14_mvec)       (parent:^w14_mvec)
    (buf1:^w32_mvec)       (buf2:^w32_mvec)
    (arg:^w32_mvec)
    =
```

```
 ? (reset:^msig)     (button:^msig)
   (atomflag:^msig)  (bit30flag:^msig) (bit31flag:^msig)
   (zeroflag:^msig)  (nilflag:^msig)   (eqflag:^msig)
   (leqflag:^msig)
   (flag0:^msig)     (flag1:^msig)
   (ralu:^msig)      (rmem:^msig)      (rarg:^msig)
   (rbuf1:^msig)     (rbuf2:^msig)     (rcar:^msig)
   (rs:^msig)        (re:^msig)        (rc:^msig)
   (rd:^msig)        (rmar:^msig)      (rx1:^msig)
   (rx2:^msig)       (rfree:^msig)     (rparent:^msig)
   (rroot:^msig)     (ry1:^msig)       (ry2:^msig)
   (rnum:^msig)      (rnil:^msig)      (rtrue:^msig)
   (rfalse:^msig)    (rcons:^msig)
   (write_bit:^msig) (bidir:^msig)
   (warg:^msig)      (wbuf1:^msig)
   (wbuf2:^msig)     (wcar:^msig)      (ws:^msig)
   (we:^msig)        (wc:^msig)        (wd:^msig)
   (wmar:^msig)      (wx1:^msig)       (wx2:^msig)
   (wfree:^msig)     (wparent:^msig)   (wroot:^msig)
   (wy1:^msig)       (wy2:^msig)
   (dec:^msig)       (add:^msig)       (sub:^msig)
   (mul:^msig)       (div:^msig)       (rem:^msig)
   (setbit30:^msig)  (setbit31:^msig)  (resetbit31:^msig)
   (replcar:^msig)   (replcdr:^msig)   (resetbit30:^msig)
   (bus_bits:^w32_mvec) (mem_bits:^w32_mvec) (mar_bits:^w14_mvec)
   .
(CU   SYS_Clocked
      reset button
      mpc s0 s1 s2 s3
      (Opcode arg)
      atomflag bit30flag bit31flag zeroflag nilflag eqflag leqflag
      flag0    flag1
      ralu   rmem   rarg
      rbuf1  rbuf2  rcar
      rs     re     rc
      rd     rmar   rx1
      rx2    rfree  rparent
      rroot  ry1    ry2
      rnum   rnil   rtrue
      rfalse rcons
      write_bit         bidir
      warg   wbuf1
      wbuf2  wcar    ws
      we     wc      wd
      wmar   wx1     wx2
      wfree  wparent wroot
      wy1    wy2
      dec      add      sub
      mul      div      rem
      setbit30 setbit31 resetbit31
      replcar  replcdr  resetbit30)
 /\
 (DP  bus_bits mem_bits SYS_Clocked
      rmem
```

```
        mar_bits wmar    rmar      rnum rnil rtrue rfalse
        s'       ws      rs
        e'       we      re
        c'       wc      rc
        d'       wd      rd
        free     wfree   rfree
        parent   wparent rparent
        root     wroot   rroot
        y1       wy1     ry1
        x1       wx1     rx1
        x2       wx2     rx2
        y2       wy2     ry2      rcons
        car      wcar    rcar
        atomflag bit30flag bit31flag zeroflag nilflag eqflag leqflag
        arg      warg    rarg
        buf1     wbuf1   rbuf1
        buf2     wbuf2   rbuf2
        replcar  replcdr sub add dec mul div rem
        setbit30 setbit31 resetbit30 resetbit31
        ralu )
    /\
    (rt_PF reset         button
           flag0         flag1
           bidir         write_bit   rmem
           bus_bits      mem_bits    mar_bits
           reset_pin     button_pin
           flag0_pin     flag1_pin
           write_bit_pin rmem_pin
           bus_pins      mar_pins)"
  );;

close_theory ();;
```

3.1.7 rt_SYS

```
% SECD verification                                                    %
%                                                                     %
% FILE:                rt_SYS.ml                                      %
%                                                                     %
% DESCRIPTION:  This describes the top level view of the SECD         %
%               system implementation, which is the SECD chip         %
%               wired to memory.                                      %
%                                                                     %
% USES FILES:   rt_SECD.th                                            %
%                                                                     %
% Brian Graham 88.08.11                                               %
%                                                                     %
% Modifications:                                                      %
% 88.??.?? - defined my own simple memory to overcome problems        %
%            with initialization.                                     %
% 88.10.07 - x1,x2,buf1 added to SECD parameters.                     %
%=====================================================================%
new_theory `rt_SYS`;;

loadf `load_wordn`;;

new_parent `rt_SECD`;;

%---------------------------------------------------------------------%
% We will not make use of the built-in memory type.                   %
% The reason is that there is no way of easily defining a             %
% startup state for the memory.                                       %
%                                                                     %
% I'm not really sure if this initialization is a problem.            %
% It may well be for inductive proofs, or for ...                     %
%---------------------------------------------------------------------%

let mtime = ":num";;

let mem14_32 = ":word14->word32";;

let msig        = ":^mtime->bool"
and w9_mvec     = ":^mtime->word9"
and w32_mvec    = ":^mtime->word32"
and w14_mvec    = ":^mtime->word14"
and m14_32_mvec = ":^mtime->^mem14_32";;

let EMPTY14_32 = new_definition
  (`EMPTY14_32`,
   "EMPTY14_32 (a:word14) = #00000000000000000000000000000000"
  );;

let Fetch14 = new_definition
  (`Fetch14`,
   "Fetch14 (mar:word14) (bus:word32) (mem:^mem14_32) = (bus = mem mar)"
  );;
```

```
let Store14 = new_definition
  ('Store14',
   "Store14 (mar:word14) (bus:word32) (mem:^mem14_32) =
     (\a. (a = mar) => bus | mem a)"
  );;

%-------------------------------------------------------------%
% This is a simple model of a Static RAM memory.  The control%
% lines are as named on the AMD Am99C88/Am99CL88 family of   %
% 8K x 8 CMOS Static Random Access Memories.                 %
% Configuration is as follows:                               %
%                W_bar         G_bar         function        %
%                 H             H            output disabled %
%                 H             L            read            %
%                 L             X            write           %
%                                                            %
% Memory is a vector, like any other signal, that changes    %
% over time.  There is an output only if both controls are   %
% asserted correctly.                                        %
%-------------------------------------------------------------%
let SRAM = new_definition
  ('SRAM',
   "SRAM mem W_bar G_bar address_in in_out =
    (!t.
        (mem(SUC t) =
           ((~W_bar t) => Store14(address_in t)(in_out t)(mem t)
                        | mem t)
        ) /\
        (W_bar t /\ ~G_bar t ==>
                          Fetch14 (address_in t)(in_out t)(mem t)))"
  );;

%-------------------------------------------------------------%
% The system consists of the chip connected to a memory.     %
%-------------------------------------------------------------%
let SYS = new_definition
  ('SYS',
   "SYS memory
       SYS_Clocked
       mpc s0  s1  s2  s3
       reset_pin       button_pin
       flag0_pin       flag1_pin
       write_bit_pin   rmem_pin
       bus_pins        mar_pins
       s'  e'  c'  d'  free
       x1  x2  y1  y2  car  root  parent
       buf1    buf2    arg  =
   (SECD     SYS_Clocked
       mpc s0  s1  s2  s3
       reset_pin       button_pin
       flag0_pin       flag1_pin
       write_bit_pin   rmem_pin
       bus_pins        mar_pins
       s'  e'  c'  d'  free
```

```
      x1  x2  y1  y2  car  root  parent
      buf1    buf2    arg) /\
   (SRAM memory write_bit_pin rmem_pin mar_pins bus_pins)"
  );;

close_theory ();;

print_theory `-`;;
```

3.2 Top Level Specification

The top level specification defines a 4-state finite state machine. From the "top_of_cycle" state, there is one transition for each machine instruction, and these are defined using abstract memory operations

3.2.1 top-SECD

```
% SECD verification                                                    %
%                                                                     %
% FILE:          top_SECD.ml                                          %
%                                                                     %
% DESCRIPTION: An attempt to define the behaviour of the SECD         %
%              system (chip plus memory), in terms of how memory      %
%              is altered by each machine instruction.                %
%                                                                     %
% USES FILES:  abstract_mem_type.th, cu_types.th, dp_types.th         %
%                                                                     %
% Brian Graham 88.08.24                                               %
%                                                                     %
% Modifications:                                                      %
% 89.10.11 - removed time parameter from top level.                   %
%                                                                     %
%=====================================================================%
new_theory `top_SECD`;;

loadf `load_wordn`;;

map new_parent [ `abstract_mem_type`
               ; `cu_types`
               ; `dp_types`
               ];;

let ftime = ":num"
and mtime = ":num"
and ctime = ":num";;

let fsig = ":^ftime->bool"
and msig = ":^mtime->bool"
and csig = ":^ctime->bool";;

let w9_fvec = ":^ftime->word9"
and w9_mvec = ":^mtime->word9"
and w9_cvec = ":^ctime->word9";;

let w14_fvec = ":^ftime->word14"
and w14_mvec = ":^mtime->word14"
and w14_cvec = ":^ctime->word14";;

let w27_fvec = ":^ftime->word27"
and w27_mvec = ":^mtime->word27"
and w27_cvec = ":^ctime->word27";;
```

```
let w32_fvec = ":^ftime->word32"
and w32_mvec = ":^mtime->word32"
and w32_cvec = ":^ctime->word32";;

load_all `abstract_mem_type`;;
%=====================================================================%
% A higher order iterative function.                                  %
%=====================================================================%
let nth = new_prim_rec_definition
  (`nth`,
  "(nth    0   (fcn:(*->*)) (x:*) = x               ) /\
   (nth (SUC n) fcn          x    = fcn (nth n fcn x))"
  );;

let LD_   = "INT 1"
and LDC_  = "INT 2"
and LDF_  = "INT 3"
and AP_   = "INT 4"
and RTN_  = "INT 5"
and DUM_  = "INT 6"
and RAP_  = "INT 7"
and SEL_  = "INT 8"
and JOIN_ = "INT 9"
and CAR_  = "INT 10"
and CDR_  = "INT 11"
and ATOM_ = "INT 12"
and CONS_ = "INT 13"
and EQ_   = "INT 14"
and ADD_  = "INT 15"
and SUB_  = "INT 16"
and MUL_  = "INT 17"
and DIV_  = "INT 18"
and REM_  = "INT 19"
and LEQ_  = "INT 20"
and STOP_ = "INT 21";;

%=====================================================================%
% Handy types for interactive use.                                    %
%=====================================================================%
let M = ":(word14,atom)mfsexp_mem";;
let M_csig = ":^ctime->^M" ;;

let S = ":(word14#word14)+atom"
and C = ":word14";;

let state = ":bool#bool";;
let state_msig = ":^mtime->^state"
and state_csig = ":^ctime->^state";;
```

```
%=====================================================================%
% State will be described by a boolean pair.                          %
% Initially, only 4 states are defined, being:                        %
%       idle                                                          %
%       top_of_cycle                                                  %
%       error0                                                        %
%       error1                                                        %
%=====================================================================%
let idle = new_definition
  (`idle`, "idle = (F,F)");;

let top_of_cycle = new_definition
  (`top_of_cycle`, "top_of_cycle = (F,T)");;

let error0 = new_definition
  (`error0`, "error0 = (T,F)");;

let error1 = new_definition
  (`error1`, "error1 = (T,T)");;

%=====================================================================%
% Extract the cell component from the triple returned from            %
% M_Cons, Car, Cdr, etc.                                              %
%=====================================================================%
let cell_of = new_definition
  (`cell_of`, "cell_of (a:*,b:**,c:***) = a");;

%=====================================================================%
% Extract the memory component from the triple returned from          %
% M_Cons, Car, Cdr, etc.                                              %
%=====================================================================%
let mem_of = new_definition
  (`mem_of`, "mem_of (a:*,b:**,c:***) = b");;

%=====================================================================%
% Extract the free component from the triple returned from            %
% M_Cons, Car, Cdr, etc.                                              %
%=====================================================================%
let free_of = new_definition
  (`free_of`, "free_of (a:*,b:**,c:***) = c");;

%=====================================================================%
% Extract the memory and free components from the triple returned     %
% from M_Cons, Car, Cdr, etc.                                         %
%=====================================================================%
let mem_free_of = new_definition
  (`mem_free_of`, "mem_free_of (a:*,b:**,c:***) = (b,c)");;
```

```
%=====================================================================%
% This should be revised to be consistent with integer theory, for%
% example if the integer is (3,1), we want num 2.                    %
%=====================================================================%
let pos_num_of = new_definition
  (`pos_num_of`,
   "pos_num_of (i:integer) = FST (REP_integer i)"
  );;

map new_definition
    [ `Mul`,  "Mul(w:^C, x:^C, y:^M, z:^C)  =  M_Dec(w,y,z)" ;
      `Div`,  "Div(w:^C, x:^C, y:^M, z:^C)  =  M_Dec(w,y,z)" ;
      `Rem`,  "Rem(w:^C, x:^C, y:^M, z:^C)  =  M_Dec(w,y,z)"
    ];;

%=====================================================================%
% The state transition definitions for each of the 21 machine        %
% instructions follows.                                              %
%=====================================================================%
%=====================================================================%
%                          n          b                              %
%       s = M_Cons (car((cdr )(car((cdr ) e))), s)                   %
%       c = cdr (cdr (c))                                            %
%=====================================================================%
let LD_trans = new_definition
(`LD_trans`,
 "LD_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let m = pos_num_of (M_int_of (M_CAR(M_CAR(M_CDR(c,MEM,free)))))
        in
        let n = pos_num_of (M_int_of (M_CDR(M_CAR(M_CDR (c,MEM,free)))))
        in
        let cell_mem_free =             % returns (cell,(MEM,free)) %
          M_Cons_tr (s,
                   M_CAR (nth n M_CDR
                            (M_CAR (nth m M_CDR
                                     (e, MEM,free)))))
        in
        (cell_of cell_mem_free,
         e,
         M_Car (M_CDR (M_CDR (c, mem_free_of cell_mem_free))),
         d,
         free_of cell_mem_free,
         mem_of cell_mem_free,
         top_of_cycle)");;
```

```
%================================================================%
%       s = M_Cons (car (cdr (c)), s)                             %
%       c = cdr (cdr (c))                                         %
%================================================================%
let LDC_trans = new_definition
(`LDC_trans`,
 "LDC_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let cell_mem_free =
            M_Cons_tr (s, M_CAR(M_CDR(c,MEM,free)))
        in
        (cell_of cell_mem_free,
         e,
         M_Cdr(M_CDR (c,mem_free_of cell_mem_free)),
         d,
         free_of cell_mem_free,
         mem_of cell_mem_free,
         top_of_cycle)");;

%================================================================%
%       s = cons (cons (car(cdr(c)),e), s)                        %
%       c = cdr (cdr (c))                                         %
%================================================================%
let LDF_trans = new_definition
(`LDF_trans`,
 "LDF_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let cell_mem_free =
            M_Cons_tr (s, M_Cons_tr (e, M_CAR(M_CDR(c,MEM,free))))
        in
        (cell_of cell_mem_free,
         e,
         M_Cdr(M_CDR (c,mem_free_of cell_mem_free)),
         d,
         free_of cell_mem_free,
         mem_of cell_mem_free,
         top_of_cycle)");;
```

```
%================================================================%
%       d = cons (cdr(cdr(s)), cons(e, cons(cdr(c),d)))          %
%       e = cons (car(cdr(s)), cdr(car(s)))                      %
%       c = car (car (s))                                        %
%       s = NIL                                                  %
% This needs some thought yet.  What is the type of Nil?         %
%================================================================%
let AP_trans = new_definition
(`AP_trans`,
 "AP_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let cell_mem_free = M_Cons(e, M_Cons_tr(d,M_CDR(c,MEM,free)))
        in
        let d_mem_free = M_Cons_tr(cell_of cell_mem_free,
                                   M_CDR(M_CDR(s,mem_free_of cell_mem_free)))
        in
        let e_mem_free = M_Cons (M_Car (M_CDR (s,mem_free_of d_mem_free)),
                                 M_CDR (M_CAR (s,mem_free_of d_mem_free)))
        in
        (NIL_addr,
         cell_of e_mem_free,
         M_Car (M_CAR (s,mem_free_of e_mem_free)),
         cell_of d_mem_free,
         free_of e_mem_free,
         mem_of e_mem_free,
         top_of_cycle)");;

%================================================================%
%       s = cons (car(s), car(d))                                %
%       e = car (cdr(d))                                         %
%       c = car (cdr (cdr(d)))                                   %
%       d = cdr (cdr (cdr(d)))                                   %
%================================================================%
let RTN_trans = new_definition
(`RTN_trans`,
 "RTN_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let s_mem_free = M_Cons (M_Car (s,MEM,free),
                                 M_CAR (d,MEM,free))
        in
        (cell_of s_mem_free,
         M_Car(M_CDR(d,mem_free_of s_mem_free)),
         M_Car(M_CDR(M_CDR(d,mem_free_of s_mem_free))),
         M_Cdr(M_CDR(M_CDR(d,mem_free_of s_mem_free))),
         free_of s_mem_free,
         mem_of  s_mem_free,
         top_of_cycle)");;
```

```
%=====================================================================%
%       e = cons (Nil, e)                                             %
%       c = cdr (c)                                                   %
%=====================================================================%
let DUM_trans = new_definition
(`DUM_trans`,
 "DUM_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let e_mem_free = M_Cons (NIL_addr,e,MEM,free)
        in
        (s,
         cell_of e_mem_free,
         M_Cdr(c,mem_free_of e_mem_free),
         d,
         free_of e_mem_free,
         mem_of  e_mem_free,
         top_of_cycle)");;

%=====================================================================%
%       d = cons (cdr(cdr(s)), cons(cdr(e), cons(cdr(c),d)))          %
%       e = M_Rplaca (cdr(car(s)), car(cdr(s)))                       %
%       c = car (car (s))                                             %
%       s = NIL                                                       %
%=====================================================================%
let RAP_trans = new_definition
(`RAP_trans`,
 "RAP_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let cell1_mem_free = M_Cons_tr(d,M_CDR(c,MEM,free))
        in
        let cell2_mem_free = M_Cons_tr(cell_of cell1_mem_free,
                                  M_CDR(e,mem_free_of cell1_mem_free))
        in
        let d_mem_free = M_Cons_tr (cell_of cell2_mem_free,
                                  M_CDR(M_CDR(s,mem_free_of cell2_mem_free)))
        in
        let e_mem_free = M_Rplaca (M_Cdr (M_CAR (s,mem_free_of d_mem_free)),
                                 M_CAR(M_CDR (s,mem_free_of d_mem_free)))
        in
        (NIL_addr,
         cell_of e_mem_free,
         M_Car(M_CAR(s,mem_free_of e_mem_free)),
         cell_of d_mem_free,
         free_of e_mem_free,
         mem_of  e_mem_free,
         top_of_cycle)");;
```

```
%================================================================%
%       d = cons (cdr(cdr(cdr(c))), d)                           %
%       c = (mem[True] = mem[car(s)]) => car(cdr(c))    |        %
%                                        car(cdr(cdr(c)))        %
%       s = cdr(s)                                               %
%================================================================%
let SEL_trans = new_definition
(`SEL_trans`,
 "SEL_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let d_mem_free = M_Cons_tr (d,M_CDR(M_CDR(M_CDR(c,MEM,free))))
        in
        let cond = M_is_T (M_CAR (s,mem_free_of d_mem_free))
        in
        (M_Cdr (s, mem_free_of d_mem_free),
         e,
         (cond => M_Car(M_CDR (c,mem_free_of d_mem_free))     |
                  M_Car(M_CDR(M_CDR(c,mem_free_of d_mem_free)))),
         cell_of d_mem_free,
         free_of d_mem_free,
         mem_of  d_mem_free,
         top_of_cycle)");;

%================================================================%
%       c = car(d)                                               %
%       d = cdr(d)                                               %
%================================================================%
let JOIN_trans = new_definition
(`JOIN_trans`,
 "JOIN_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        (s,
         e,
         M_Car(d,MEM,free),
         M_Cdr(d,MEM,free),
         free,
         MEM,
         top_of_cycle)");;

%================================================================%
%       s = cons (car(car(s)), cdr(s))                           %
%       c = cdr(c)                                               %
%================================================================%
let CAR_trans = new_definition
(`CAR_trans`,
 "CAR_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let s_mem_free = M_Cons (M_Car(M_CAR (s,MEM,free)),
                                 M_CDR(s,MEM,free))
        in
        (cell_of s_mem_free,
         e,
         M_Cdr(c,mem_free_of s_mem_free),
         d,
         free_of s_mem_free,
         mem_of  s_mem_free,
         top_of_cycle)");;
```

```
%=====================================================================%
%       s = cons (cdr(car(s)), cdr(s))                                %
%       c = cdr(c)                                                    %
%=====================================================================%
let CDR_trans = new_definition
(`CDR_trans`,
 "CDR_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let s_mem_free = M_Cons (M_Cdr(M_CAR (s,MEM,free)),
                                 M_CDR(s,MEM,free))
        in
        (cell_of s_mem_free,
         e,
         M_Cdr(c,mem_free_of s_mem_free),
         d,
         free_of s_mem_free,
         mem_of  s_mem_free,
         top_of_cycle)");;

%=====================================================================%
%       s = cons ((ATOM(mem[car(s)]) => True | False, cdr(s))         %
%       c = cdr(c)                                                    %
%=====================================================================%
let ATOM_trans = new_definition
(`ATOM_trans`,
 "ATOM_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let s_mem_free = M_Cons ((M_Atom (M_CAR (s,MEM,free))=>
                                          T_addr|F_addr),
                                 M_CDR(s,MEM,free))
        in
        (cell_of s_mem_free,
         e,
         M_Cdr(c,mem_free_of s_mem_free),
         d,
         free_of s_mem_free,
         mem_of  s_mem_free,
         top_of_cycle)");;
```

```
%================================================================%
%       s = cons (cons (car(s), car(cdr(s))), cdr(cdr(s)))       %
%       c = cdr (c)                                              %
%================================================================%
let CONS_trans = new_definition
(`CONS_trans`,
 "CONS_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let cell_mem_free = M_Cons (M_Car (s,MEM,free),
                                    M_CAR(M_CDR(s,MEM,free)))
        in
        let s_mem_free = M_Cons(cell_of cell_mem_free,
                            M_CDR(M_CDR(s,mem_free_of cell_mem_free)))
        in
        (cell_of s_mem_free,
         e,
         M_Cdr(c,mem_free_of s_mem_free),
         d,
         free_of s_mem_free,
         mem_of  s_mem_free,
         top_of_cycle)");;

%================================================================%
%       s = cons ((mem[car(cdr(s))]=mem[car(s)]) =>               %
%                       True | False, cdr(cdr(s)))               %
%       c = cdr(c)                                               %
%================================================================%
let EQ_trans = new_definition
(`EQ_trans`,
 "EQ_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let s_mem_free = M_Cons ((M_Eq(M_Car(M_CDR(s,MEM,free)),
                                   M_CAR(s,MEM,free))=>T_addr|F_addr),
                             M_CDR(M_CDR(s,MEM,free)))
        in
        (cell_of s_mem_free,
         e,
         M_Cdr(c,mem_free_of s_mem_free),
         d,
         free_of s_mem_free,
         mem_of  s_mem_free,
         top_of_cycle)");;
```

```
%=====================================================================%
%       s = cons (M_Add (mem[car(cdr(s))],                             %
%                        mem[car(s)]), cdr(cdr(s)))                    %
%       c = cdr (c)                                                    %
%=====================================================================%
let ADD_trans = new_definition
(`ADD_trans`,
 "ADD_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let arg1 = M_Car(M_CDR(s,MEM,free))
        in
        let arg2 = M_Car(s,MEM,free)
        in
        let cell_mem_free = M_Add (arg1,arg2,MEM,free)
        in
        let s_mem_free = M_Cons (cell_of cell_mem_free,
                                 M_CDR(M_CDR(s,mem_free_of cell_mem_free)))
        in
        (cell_of s_mem_free,
         e,
         M_Cdr(c,mem_free_of s_mem_free),
         d,
         free_of s_mem_free,
         mem_of  s_mem_free,
         top_of_cycle)");;

%=====================================================================%
%       s = cons (M_Sub (mem[car(cdr(s))],                             %
%                        mem[car(s)]), cdr(cdr(s)))                    %
%       c = cdr (c)                                                    %
%=====================================================================%
let SUB_trans = new_definition
(`SUB_trans`,
 "SUB_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let arg1 = M_Car(M_CDR(s,MEM,free))
        in
        let arg2 = M_Car(s,MEM,free)
        in
        let cell_mem_free = M_Sub (arg1,arg2,MEM,free)
        in
        let s_mem_free = M_Cons (cell_of cell_mem_free,
                                 M_CDR(M_CDR(s,mem_free_of cell_mem_free)))
        in
        (cell_of s_mem_free,
         e,
         M_Cdr(c,mem_free_of s_mem_free),
         d,
         free_of s_mem_free,
         mem_of  s_mem_free,
         top_of_cycle)");;
```

```
%=====================================================================%
%       s = cons (Mul (mem[car(cdr(s))],                              %
%                      mem[car(s)]), cdr(cdr(s)))                     %
%       c = cdr (c)                                                   %
%=====================================================================%
let MUL_trans = new_definition
('MUL_trans',
 "MUL_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let arg1 = M_Car(M_CDR(s,MEM,free))
        in
        let arg2 = M_Car(s,MEM,free)
        in
        let cell_mem_free = Mul (arg1,arg2,MEM,free)
        in
        let s_mem_free = M_Cons (cell_of cell_mem_free,
                                 M_CDR(M_CDR(s,mem_free_of cell_mem_free)))
        in
        (cell_of s_mem_free,
         e,
         M_Cdr(c,mem_free_of s_mem_free),
         d,
         free_of s_mem_free,
         mem_of  s_mem_free,
         top_of_cycle)");;

%=====================================================================%
%       s = cons (Div (mem[car(cdr(s))],                              %
%                      mem[car(s)]), cdr(cdr(s)))                     %
%       c = cdr (c)                                                   %
%=====================================================================%
let DIV_trans = new_definition
('DIV_trans',
 "DIV_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let arg1 = M_Car(M_CDR(s,MEM,free))
        in
        let arg2 = M_Car(s,MEM,free)
        in
        let cell_mem_free = Div (arg1,arg2,MEM,free)
        in
        let s_mem_free = M_Cons (cell_of cell_mem_free,
                                 M_CDR(M_CDR(s,mem_free_of cell_mem_free)))
        in
        (cell_of s_mem_free,
         e,
         M_Cdr(c,mem_free_of s_mem_free),
         d,
         free_of s_mem_free,
         mem_of  s_mem_free,
         top_of_cycle)");;
```

```
%================================================================%
%       s = cons (Rem (mem[car(cdr(s))],                         %
%                      mem[car(s)]), cdr(cdr(s)))                %
%       c = cdr (c)                                              %
%================================================================%
let REM_trans = new_definition
('REM_trans',
 "REM_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let arg1 = M_Car(M_CDR(s,MEM,free))
        in
        let arg2 = M_Car(s,MEM,free)
        in
        let cell_mem_free = Rem (arg1,arg2,MEM,free)
        in
        let s_mem_free = M_Cons (cell_of cell_mem_free,
                                 M_CDR(M_CDR(s,mem_free_of cell_mem_free)))
        in
        (cell_of s_mem_free,
         e,
         M_Cdr(c,mem_free_of s_mem_free),
         d,
         free_of s_mem_free,
         mem_of  s_mem_free,
         top_of_cycle)");;

%================================================================%
%       s = cons ((mem[car(cdr(s))]<=mem[car(s)]) =>             %
%                        True|False, cdr(cdr(s)))                %
%       c = cdr (c)                                              %
%================================================================%
let LEQ_trans = new_definition
('LEQ_trans',
 "LEQ_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let arg1 = M_Car(M_CDR(s,MEM,free))
        in
        let arg2 = M_Car(s,MEM,free)
        in
        let s_mem_free = M_Cons ((M_Leq (arg1,arg2,MEM,free)=>
                                      T_addr|F_addr),
                                 M_CDR(M_CDR(s,MEM,free)))
        in
        (cell_of s_mem_free,
         e,
         M_Cdr(c,mem_free_of s_mem_free),
         d,
         free_of s_mem_free,
         mem_of  s_mem_free,
         top_of_cycle)");;
```

```
%================================================================%
%       s = car(s)                                               %
%       cdr(mem[num]) = car(s)                                   %
%    NOTE: the car bits are undefined value                      %
%================================================================%
let STOP_trans = new_definition
('STOP_trans',
 "STOP_trans (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
        let new_s = M_Car (s,MEM,free)
        in
        (new_s,
         e,
         c,
         d,
         free,
         (@m. REP_mfsexp_mem m =
            (\a. (a = NUM_addr) => (F,F), INL (@v. SND v = new_s) |
                                  REP_mfsexp_mem MEM a)),
         idle)");;

%================================================================%
% Returns the 4 main registers, free pointer, memory, plus a     %
% state value that relates to the fsm.                           %
% It expects the instruction value to be in the range:           %
%       INT 1 to INT 21.                                         %
%================================================================%
let NEXT = new_definition
('NEXT',
 "NEXT (s:^C,e:^C,c:^C,d:^C,free:^C,MEM:^M) =
  let instr = M_int_of (M_CAR (c, MEM, free))
  in
  ((instr = ^LD_)   =>   (LD_trans   (s,e,c,d,free,MEM)) |
   (instr = ^LDC_)  =>   (LDC_trans  (s,e,c,d,free,MEM)) |
   (instr = ^LDF_)  =>   (LDF_trans  (s,e,c,d,free,MEM)) |
   (instr = ^AP_)   =>   (AP_trans   (s,e,c,d,free,MEM)) |
   (instr = ^RTN_)  =>   (RTN_trans  (s,e,c,d,free,MEM)) |
   (instr = ^DUM_)  =>   (DUM_trans  (s,e,c,d,free,MEM)) |
   (instr = ^RAP_)  =>   (RAP_trans  (s,e,c,d,free,MEM)) |
   (instr = ^SEL_)  =>   (SEL_trans  (s,e,c,d,free,MEM)) |
   (instr = ^JOIN_) =>   (JOIN_trans (s,e,c,d,free,MEM)) |
   (instr = ^CAR_)  =>   (CAR_trans  (s,e,c,d,free,MEM)) |
   (instr = ^CDR_)  =>   (CDR_trans  (s,e,c,d,free,MEM)) |
   (instr = ^ATOM_) =>   (ATOM_trans (s,e,c,d,free,MEM)) |
   (instr = ^CONS_) =>   (CONS_trans (s,e,c,d,free,MEM)) |
   (instr = ^EQ_)   =>   (EQ_trans   (s,e,c,d,free,MEM)) |
   (instr = ^ADD_)  =>   (ADD_trans  (s,e,c,d,free,MEM)) |
   (instr = ^SUB_)  =>   (SUB_trans  (s,e,c,d,free,MEM)) |
   (instr = ^MUL_)  =>   (MUL_trans  (s,e,c,d,free,MEM)) |
   (instr = ^DIV_)  =>   (DIV_trans  (s,e,c,d,free,MEM)) |
   (instr = ^REM_)  =>   (REM_trans  (s,e,c,d,free,MEM)) |
   (instr = ^LEQ_)  =>   (LEQ_trans  (s,e,c,d,free,MEM)) |
%  (instr = ^STOP_) %    (STOP_trans (s,e,c,d,free,MEM))
  )"
);;
```

```
%================================================================%
% The clock signal is not used, so is omitted for now.  If I     %
% want to extend the behavioural description to include shift    %
% registers, then both clock signals will need to be added       %
% (or use the clock and its inverse).  They should be abstracted %
% from rt level such that the rt_SYS_clocked signal holds for    %
% every point in the interval between points of ctime.  Also     %
% required will be a change to the abstraction points of mtime,  %
% to allow some points when the SR_clocks operate.               %
%                                                                %
% Note that the reset input has been eliminated already, since it %
% is asserted when we may not be in a major state.  The top level %
% specification will only hold if we are properly initialized...  %
%                                                                %
% 89.08.30 - reintroduced a parameter "t", so that the           %
%            constraints in the proof statement can limit us to  %
%            those times when we are in a major state, rather    %
%            than trying to prove the spec holds for times       %
%            beginning at other states.                          %
% 89.10.11 - eliminated t parameter again - clearer thinking     %
%            prevails.                                           %
%================================================================%
let SYS_spec = new_definition
(`SYS_spec`,
 "SYS_spec (MEM:^M_csig)
      %    (SYS_Clocked:^csig)             omitted for now   %
           (s:^w14_cvec) (e:^w14_cvec) (c:^w14_cvec) (d:^w14_cvec)
           (free:^w14_cvec)
           (button_pin:^csig)
           (state:^state_csig)
  =

  (state 0 = idle) /\

 !t:^ctime.
  ((s (SUC t),    e (SUC t),   c (SUC t),     d (SUC t),
    free (SUC t), MEM (SUC t), state (SUC t)) =
   (state t = idle) =>
    (button_pin t => (M_Cdr (M_CAR (NUM_addr, MEM t, free t)),
                      NIL_addr,
                      M_Car (M_CDR (NUM_addr, MEM t, free t)),
                      NIL_addr, NIL_addr, MEM t, top_of_cycle) |
                     (s t, e t, c t, d t, free t, MEM t, idle)) |
   (state t = error0) =>
    (button_pin t => (s t, e t, c t, d t, free t, MEM t, error1) |
                     (s t, e t, c t, d t, free t, MEM t, error0)) |
   (state t = error1) =>
    (button_pin t => (s t, e t, c t, d t, free t, MEM t, error1) |
                     (s t, e t, c t, d t, free t, MEM t, idle)) |
 % (state = top_of_cycle) %
   (NEXT (s t, e t, c t, d t, free t, MEM t)))");;

close_theory ();;
print_theory `-`;;
```

Chapter 4

Abstractions and Proofs

4.1 Constraining the Problem

The SECD chip was designed to operate under rather specific constraints, concerning the operation of the clocks, the use of the reset signal to give a deterministic startup state, input signal stability, etc. Unstated constraints are inherent in the model used to define the circuit, including such things as voltage levels, the clocking speed which must permit all circuit points settle to a stable value before any clock transition, the existence of an external memory that operates according to its definition, and so on.

We constrain the reset signal operation and the clock operation specifically. It should be noted that the existence of a clock signal at the register transfer level is unusual; normally this granularity of time advances directly with the clock. The existence of a distinct clock for shiftregister operation on SECD necessitates the appearance of the system clock in this definition. Additionally, we limit the problem to never exhaust available memory. Eventually, we wish to include a proof of correctness of the garbage collector, but the issue of a problem size constraint when proving correctness of a finite size machine is a nontrivial issue which is the subject of some concern in any verification effort.

Finally, we wish to be concerned only with system operation when given valid programs. The partially complete specification here requires extension to instruction arguments as well. It is hoped that a proof of correctness of the compiler will fulfill the requirements of this last constraint.

4.1.1 constraints

```
% SECD verification                                                    %
%                                                                     %
% FILE:                constraints.ml                                 %
%                                                                     %
% DESCRIPTION:  Contains the constraint predicates that limit the     %
%               states that are subject to verification.              %
%                                                                     %
% USES FILES:   rt_SYS.th, top_SECD.th                                %
%                                                                     %
% Brian Graham 89.09.11                                               %
%                                                                     %
% Modifications:                                                      %
%                                                                     %
%=====================================================================%
new_theory 'constraints';;
```

```
loadf `load_wordn`;;

map new_parent [`rt_SYS` ;
                `top_SECD` ];;

let ftime = ":num"
and mtime = ":num"
and ctime = ":num";;

let fsig = ":^ftime->bool"
and msig = ":^mtime->bool"
and csig = ":^ctime->bool";;

let w9_fvec = ":^ftime->word9"
and w9_mvec = ":^mtime->word9"
and w9_cvec = ":^ctime->word9";;

let w14_fvec = ":^ftime->word14"
and w14_mvec = ":^mtime->word14"
and w14_cvec = ":^ctime->word14";;

let w27_fvec = ":^ftime->word27"
and w27_mvec = ":^mtime->word27"
and w27_cvec = ":^ctime->word27";;

let w32_fvec = ":^ftime->word32"
and w32_mvec = ":^mtime->word32"
and w32_cvec = ":^ctime->word32";;

let mem14_32 = ":word14->word32";;
let m14_32_mvec = ":^mtime->^mem14_32";;
let M = ":(word14,atom)mfsexp_mem";;
let M_mvec = ":^mtime->^M";;

let state = ":bool # bool";;
let state_msig = ":^mtime->^state"
and state_csig = ":^ctime->^state";;

let LD_   = "INT 1"
and LDC_  = "INT 2"
and LDF_  = "INT 3"
and AP_   = "INT 4"
and RTN_  = "INT 5"
and DUM_  = "INT 6"
and RAP_  = "INT 7"
and SEL_  = "INT 8"
and JOIN_ = "INT 9"
and CAR_  = "INT 10"
and CDR_  = "INT 11"
and ATOM_ = "INT 12"
and CONS_ = "INT 13"
and EQ_   = "INT 14"
and ADD_  = "INT 15"
```

```
and SUB_  = "INT 16"
and MUL_  = "INT 17"
and DIV_  = "INT 18"
and REM_  = "INT 19"
and LEQ_  = "INT 20"
and STOP_ = "INT 21";;

%================================================================%
% Abstracting from the "state" of the rt level implementation to %
% the top level spec state concerns only the mpc contents.  The  %
% mapping is defined only for 4 mpc values, and is an arbitrary  %
% value otherwise.                                               %
%================================================================%
let state_abs = new_definition
 (`state_abs`,
  "state_abs mpc =
      (mpc = #000010110) => idle          |  % 22 %
      (mpc = #000011000) => error0        |  % 24 %
      (mpc = #000011010) => error1        |  % 26 %
      (mpc = #000101011) => top_of_cycle  |  % 43 %
                            (@stat.F)");;

let state_abs_thm = prove_thm
(`state_abs_thm`,
 "(state_abs #000010110 = idle)         /\
  (state_abs #000011000 = error0)       /\
  (state_abs #000101011 = top_of_cycle) /\
  (state_abs #000011010 = error1)",
  rt[state_abs]
  THEN in_conv_tac wordn_CONV
  THEN rt[theorem `cu_types` `Word9_11`; Bus_11; Wire_11]
  );;

let is_major_state = new_definition
 (`is_major_state`,
  "is_major_state mpc (t:^mtime) =
      (mpc t = #000010110) \/
      (mpc t = #000011000) \/
      (mpc t = #000101011) \/
      (mpc t = #000011010)");;

let is_major_state_lemma = prove_thm
(`is_major_state_lemma`,
 "is_major_state mpc t ==>
     (state_abs (mpc t) =
      (mpc t = #000010110) => idle          |     % 22 %
      (mpc t = #000011000) => error0        |     % 24 %
      (mpc t = #000011010) => error1        |     % 26 %
   %  (mpc t = #000101011) => % top_of_cycle) "   % 43 %  ,
 prt[is_major_state; state_abs]
 THEN STRIP_TAC
 THEN art[]
 );;
```

```
%=====================================================================%
% The reset input is asserted at mtime 0 and never reasserted.        %
%=====================================================================%
let reset_constraint = new_definition
(`reset_constraint`,
 "reset_constraint (reset_pin:^msig) =
  (~reset_pin 0) /\ (!t:^mtime. reset_pin (SUC t))"
 );;

%=====================================================================%
% The system clock always cycles, never the shift register clock.     %
%=====================================================================%
let clock_constraint = new_definition
(`clock_constraint`,
 "clock_constraint (SYS_Clocked:^msig) =
  !t:^mtime. SYS_Clocked t"
 );;

%=====================================================================%
% Once the machine is initialized, the free list is never empty.      %
%=====================================================================%
let free_list_constraint = new_definition
(`free_list_constraint`,
 "free_list_constraint (mpc:^w9_mvec) (free:^w14_mvec) =
  !t:^mtime. (state_abs (mpc t) = top_of_cycle) ==>
    !t':^mtime. (t <= t') ==> ~(free t' = NIL_addr)"
 );;

%=====================================================================%
% Once initialized, there is always a valid program in memory.        %
% Presently it is written to ensure that a valid instruction          %
% opcode is in place.  This will need revision to include a           %
% requirement for valid arguments as well.                            %
%=====================================================================%
let valid_program_constraint = new_definition
(`valid_program_constraint`,
 "valid_program_constraint (memory:^m14_32_mvec)
                           (mpc:^w9_mvec) (c':^w14_mvec) =
   !t:^mtime.
   (state_abs (mpc t) = top_of_cycle) ==>
    !t':^mtime.
     t <= t' ==>
     let instr = memory t' (c' t')
     in
     ((rec_type_bits instr = RT_CONS) /\
      let instr' = (memory t') (car_bits instr)
      in
      ((rec_type_bits instr' = RT_NUMBER) /\
       let int_instr = iVal(Bits28(atom_bits instr))
       in
        ((int_instr = ^LD_) \/
         (int_instr = ^LDC_) \/
         (int_instr = ^LDF_) \/
         (int_instr = ^AP_) \/
```

```
        (int_instr = ^RTN_) \/
        (int_instr = ^DUM_) \/
        (int_instr = ^RAP_) \/
        (int_instr = ^SEL_) \/
        (int_instr = ^JOIN_) \/
        (int_instr = ^CAR_) \/
        (int_instr = ^CDR_) \/
        (int_instr = ^ATOM_) \/
        (int_instr = ^CONS_) \/
        (int_instr = ^EQ_) \/
        (int_instr = ^ADD_) \/
        (int_instr = ^SUB_) \/
%        (int_instr = ^MUL_) \/  %
%        (int_instr = ^DIV_) \/  %
%        (int_instr = ^REM_) \/  %
        (int_instr = ^LEQ_) \/
        (int_instr = ^STOP_))))"
 );;


close_theory ();;
print_theory `-`;;
```

4.2 RTL Subcomponent Proofs

The complexity of the SECD machine reflected by the size of the RTL definition causes some considerable problem in tackling proofs. The large terms can quickly exhaust available memory when typical techniques such as rewriting are used. This has encouraged a lot of attention to the efficiency of proof, demonstrated clearly by the proofs of the control unit.

The control unit has a fairly complex definition, with 5 state values, and roughly 60 outputs. It was necessary to derive a simplified equation for each state and output, for every one of the 400 possible values in the mpc (micro program counter). Simplifying the large expression was simply not feasible, so a base theorem was proven, which unwound the definition as far as possible without a specified mpc value. Following this, the state and output equations were split into sets of related values, and exhaustive case lemmas were proven for these. Finally, a proof function can return a complete simplified expression for state and output values given any mpc value. Not only does this avoid memory exhaustion, but enables higher level proof to be tried in a reasonably interactive fashion.

4.2.1 Inc9_proofs

```
% SECD verification                                                    %
%                                                                     %
% FILE:        Inc9_proofs.ml                                         %
%                                                                     %
% DESCRIPTION: Proves a complete set of theorems about the Inc9       %
%              function applied to any relevant mpc value.            %
%                                                                     %
% USES FILES:  cu_types.th                                            %
%                                                                     %
% Brian Graham 89.09.01                                               %
%                                                                     %
% Modifications:                                                      %
%                                                                     %
%=====================================================================%
new_theory `Inc9_proofs`;;

loadf `load_wordn`;;

new_parent `cu_types`;;

let Inc9        = definition `cu_types` `Inc9`
and inc         = definition `cu_types` `inc`
and Bits9_Word9 = theorem    `cu_types` `Bits9_Word9`;;

% ================================================================== %
% ID_THM = |- !x. (\x. x)x = x                                       %
% ================================================================== %
let ID_THM =
  GEN_ALL (RIGHT_BETA (REFL "(\x:*.x)x"));;

% ================================================================== %
% We prove 9 lemmas, one for each extent that the carry propagates.  %
% This will vastly speed up the proof of the 400 lemmas, since       %
% we will need only do a wordn_CONV and a rewrite with the correct   %
```

```
% lemma.                                                                  %
% ======================================================================== %

let lemma_00 = prove_thm
  (`lemma_00`,
   "Inc9 (Word9 (Bus b9 (Bus b8 (Bus b7 (Bus b6 (Bus b5
                (Bus b4 (Bus b3 (Bus b2 (Wire F)))))))))) =
         Word9 (Bus b9 (Bus b8 (Bus b7 (Bus b6 (Bus b5
                (Bus b4 (Bus b3 (Bus b2 (Wire T)))))))))",
   prt[Inc9; o_THM; theorem `cu_types` `Bits9_Word9`; inc]
   THEN prt [LET_DEF]
   THEN re_conv_tac ETA_CONV
   THEN rt [ID_THM]
   THEN re_conv_tac (REWRITE_CONV UNCURRY_DEF ORELSEC BETA_CONV)
   THEN rt[]
  );;
```

```
let lemma_01 = prove_thm
  (`lemma_01`,
   "Inc9 (Word9 (Bus b9 (Bus b8 (Bus b7 (Bus b6 (Bus b5
                (Bus b4 (Bus b3 (Bus F (Wire T)))))))))) =
         Word9 (Bus b9 (Bus b8 (Bus b7 (Bus b6 (Bus b5
                (Bus b4 (Bus b3 (Bus T (Wire F)))))))))",
   prt[Inc9; o_THM; theorem `cu_types` `Bits9_Word9`; inc]
   THEN prt [LET_DEF]
   THEN re_conv_tac ETA_CONV
   THEN rt [ID_THM]
   THEN re_conv_tac (REWRITE_CONV UNCURRY_DEF ORELSEC BETA_CONV)
   THEN rt[]
  );;

let lemma_02 = prove_thm
  (`lemma_02`,
   "Inc9 (Word9 (Bus b9 (Bus b8 (Bus b7 (Bus b6 (Bus b5
                (Bus b4 (Bus F  (Bus T (Wire T)))))))))) =
         Word9 (Bus b9 (Bus b8 (Bus b7 (Bus b6 (Bus b5
                (Bus b4 (Bus T (Bus F (Wire F)))))))))",
   prt[Inc9; o_THM; theorem `cu_types` `Bits9_Word9`; inc]
   THEN prt [LET_DEF]
   THEN re_conv_tac ETA_CONV
   THEN rt [ID_THM]
   THEN re_conv_tac (REWRITE_CONV UNCURRY_DEF ORELSEC BETA_CONV)
   THEN rt[]
  );;

let lemma_03 = prove_thm
  (`lemma_03`,
   "Inc9 (Word9 (Bus b9 (Bus b8 (Bus b7 (Bus b6 (Bus b5
                (Bus F (Bus T  (Bus T (Wire T)))))))))) =
         Word9 (Bus b9 (Bus b8 (Bus b7 (Bus b6 (Bus b5
                (Bus T (Bus F (Bus F (Wire F)))))))))",
   prt[Inc9; o_THM; theorem `cu_types` `Bits9_Word9`; inc]
   THEN prt [LET_DEF]
   THEN re_conv_tac ETA_CONV
   THEN rt [ID_THM]
   THEN re_conv_tac (REWRITE_CONV UNCURRY_DEF ORELSEC BETA_CONV)
   THEN rt[]
  );;

let lemma_04 = prove_thm
  (`lemma_04`,
   "Inc9 (Word9 (Bus b9 (Bus b8 (Bus b7 (Bus b6 (Bus F
                (Bus T (Bus T  (Bus T (Wire T)))))))))) =
         Word9 (Bus b9 (Bus b8 (Bus b7 (Bus b6 (Bus T
                (Bus F (Bus F (Bus F (Wire F)))))))))",
   prt[Inc9; o_THM; theorem `cu_types` `Bits9_Word9`; inc]
   THEN prt [LET_DEF]
   THEN re_conv_tac ETA_CONV
   THEN rt [ID_THM]
   THEN re_conv_tac (REWRITE_CONV UNCURRY_DEF ORELSEC BETA_CONV)
   THEN rt[]);;
```

```
let lemma_05 = prove_thm
  (`lemma_05`,
   "Inc9 (Word9 (Bus b9 (Bus b8 (Bus b7 (Bus F (Bus T
                (Bus T (Bus T  (Bus T (Wire T))))))))))) =
         Word9 (Bus b9 (Bus b8 (Bus b7 (Bus T (Bus F
                (Bus F (Bus F (Bus F (Wire F))))))))))",
   prt[Inc9; o_THM; theorem `cu_types` `Bits9_Word9`; inc]
   THEN prt [LET_DEF]
   THEN re_conv_tac ETA_CONV
   THEN rt [ID_THM]
   THEN re_conv_tac (REWRITE_CONV UNCURRY_DEF ORELSEC BETA_CONV)
   THEN rt[]
  );;

let lemma_06 = prove_thm
  (`lemma_06`,
   "Inc9 (Word9 (Bus b9 (Bus b8 (Bus F (Bus T (Bus T
                (Bus T (Bus T  (Bus T (Wire T))))))))))) =
         Word9 (Bus b9 (Bus b8 (Bus T (Bus F (Bus F
                (Bus F (Bus F (Bus F (Wire F))))))))))",
   prt[Inc9; o_THM; theorem `cu_types` `Bits9_Word9`; inc]
   THEN prt [LET_DEF]
   THEN re_conv_tac ETA_CONV
   THEN rt [ID_THM]
   THEN re_conv_tac (REWRITE_CONV UNCURRY_DEF ORELSEC BETA_CONV)
   THEN rt[]
  );;

let lemma_07 = prove_thm
  (`lemma_07`,
   "Inc9 (Word9 (Bus b9 (Bus F (Bus T (Bus T (Bus T
                (Bus T (Bus T  (Bus T (Wire T))))))))))) =
         Word9 (Bus b9 (Bus T (Bus F (Bus F (Bus F
                (Bus F (Bus F (Bus F (Wire F))))))))))",
   prt[Inc9; o_THM; theorem `cu_types` `Bits9_Word9`; inc]
   THEN prt [LET_DEF]
   THEN re_conv_tac ETA_CONV
   THEN rt [ID_THM]
   THEN re_conv_tac (REWRITE_CONV UNCURRY_DEF ORELSEC BETA_CONV)
   THEN rt[]
  );;

let lemma_08 = prove_thm
  (`lemma_08`,
   "Inc9 (Word9 (Bus F (Bus T (Bus T (Bus T (Bus T
                (Bus T (Bus T  (Bus T (Wire T))))))))))) =
         Word9 (Bus T (Bus F (Bus F (Bus F (Bus F
                (Bus F (Bus F (Bus F (Wire F))))))))))",
   prt[Inc9; o_THM; theorem `cu_types` `Bits9_Word9`; inc]
   THEN prt [LET_DEF]
   THEN re_conv_tac ETA_CONV
   THEN rt [ID_THM]
   THEN re_conv_tac (REWRITE_CONV UNCURRY_DEF ORELSEC BETA_CONV)
   THEN rt[]);;
```

```
%  test_proof ...

g "Inc9 #000000000 = #000000001";;

 in_conv_tac wordn_CONV
 THEN prt[Inc9; o_THM; theorem `cu_types` `Bits9_Word9`; inc]
 THEN prt [LET_DEF]
 THEN re_conv_tac ETA_CONV
 THEN rt [ID_THM]
 THEN re_conv_tac (REWRITE_CONV UNCURRY_DEF ORELSEC BETA_CONV)
 THEN rt[]
%

let rem (x,y) = x - y * (x / y);;

letrec mk_bit_list (num, size) res =
   (size = 0) => res | (mk_bit_list ((num/2), (size-1))
                          (((string_of_int o rem) (num,2)) . res));;

let mk_word9_from_num n =
    mk_const (implode (`#` . (mk_bit_list (n,9) ([]:(string)list))),
              ":word9");;

letrec num_carries num  =
   (rem (num,2) = 0) => 0 | 1 + (num_carries (num/2));;

letrec prove_inc_thms start stop =
 (start < stop) =>
   let base = mk_word9_from_num start
   and next = mk_word9_from_num (start + 1)
   and lemm = theorem `-` (`lemma_0`^(string_of_int (num_carries start)))
   in
   ( prove_thm (concat `Inc9_lem_` (string_of_int start),
                "Inc9 ^base = ^next",
                in_conv_tac wordn_CONV
                THEN rt[lemm]
               )
   ; prove_inc_thms (start+1) stop
   )
                | ();;

prove_inc_thms 0 400;;

close_theory();;

print_theory `-`;;
```

4.2.2 CU_wordn_proofs

```
%                                                                      %
% FILE: CU_wordn_proofs.ml                                             %
%                                                                      %
% DESCRIPTION: This proves some equalities and inequalities of         %
%              wordn constants.                                        %
%                                                                      %
% USES FILES:  cu_types.th                                             %
%                                                                      %
% Brian Graham 89.09.14                                                %
%                                                                      %
% Modifications:                                                       %
%                                                                      %
%======================================================================%

new_theory `CU_wordn_proofs`;;

loadf `load_wordn`;;

new_parent `cu_types`;;

let word4_EQ_CONV = wordn_EQ_CONV (theorem `cu_types` `Word4_11`);;
let word5_EQ_CONV = wordn_EQ_CONV (theorem `cu_types` `Word5_11`);;

%----------------------------------------------------------------------%
% First, some theorems about the ineq/equality of wordn values         %
%----------------------------------------------------------------------%
letrec mk_word4_thms l n =
(0 < n) =>
 (let h' = el n l
  in
  let t' = filter (\t.not (t = h')) l
  in
  let tml = map (\t."(^h' = ^t)") t'
  in
  save_thm (`word_`^((implode o tl o explode o fst o dest_const)h'),
                 (LIST_CONJ
                  ((EQT_INTRO(REFL h')).(map word4_EQ_CONV tml))));
  mk_word4_thms l (n-1))
                 |
  TRUTH;;

letrec mk_word5_thms l n =
(0 < n) =>
 (let h' = el n l
  in
  let t' = filter (\t.not (t = h')) l
  in
  let tml = map (\t."(^h' = ^t)") t'
  in
  save_thm (`word_`^((implode o tl o explode o fst o dest_const)h'),
                 (LIST_CONJ
                  ((EQT_INTRO(REFL h')).(map word5_EQ_CONV tml))));
```

```
  mk_word5_thms l (n-1))
                |
  TRUTH;;

let l4 =
["#0000";
 "#0001";
 "#0010";
 "#0011";
 "#0100";
 "#0101";
 "#0110";
 "#0111";
 "#1000";
 "#1001";
 "#1010";
 "#1011";
 "#1100"]
and l5 =
["#00000";
 "#00001";
 "#00010";
 "#00011";
 "#00100";
 "#00101";
 "#00110";
 "#00111";
 "#01000";
 "#01001";
 "#01010";
 "#01011";
 "#01100";
 "#01101";
 "#01110";
 "#01111";
 "#10000";
 "#10001";
 "#10010";
 "#10011";
 "#10100";
 "#10101";
 "#10110";
 "#10111"];;

%-------------------------------------------------------------%
% This may benefit from changing the recursion, so memory is %
% not exhausted as quickly.                                   %
%-------------------------------------------------------------%
mk_word4_thms l4 (length l4); mk_word5_thms l5 (length l5);;

close_theory ();;
print_theory `-`;;
```

4.2.3 CU_proofs

```
%                                                                      %
% FILE: CU_proofs.ml                                                   %
%                                                                      %
% DESCRIPTION: This proves some properties of the register-            %
%              transfer definition of the control unit.                %
%                                                                      %
% USES FILES:  rt_CU.th, interface.th, constraints.th,                 %
%              Inc9_proofs.th, CU_wordn_proofs.th                      %
%                                                                      %
% Brian Graham 89.09.12                                               %
%                                                                      %
% Modifications:                                                       %
% 89.09.13 - installed functions to prove 400 lemmas for each         %
%            mpc value                                                 %
%          - also proved 37 lemmas about word4 and word5 to speed      %
%            the above proofs                                          %
% 89.09.14 - memory exhausted failure - this is too large to work.     %
%                                                                      %
%======================================================================%

new_theory `CU_proofs`;;

loadf `load_wordn`;;

map new_parent [`rt_CU`; `interface`; `constraints`; `Inc9_proofs`;
                `CU_wordn_proofs`];;

load_definition `interface` `one_asserted_12`;;

let ftime = ":num"
and mtime = ":num"
and ctime = ":num";;

let fsig = ":^ftime->bool"
and msig = ":^mtime->bool"
and csig = ":^ctime->bool";;

let w9_fvec = ":^ftime->word9"
and w9_mvec = ":^mtime->word9"
and w9_cvec = ":^ctime->word9";;

let w27_fvec = ":^ftime->word27"
and w27_mvec = ":^mtime->word27"
and w27_cvec = ":^ctime->word27";;

let fvec = ":num->^fsig"
and mvec = ":num->^msig"
and cvec = ":num->^csig";;

%----------------------------------------------------------------------%
% unwind the definition, get rid of hidden lines, etc.                 %
%----------------------------------------------------------------------%
```

```
load_library `unwind`;;

let ID_THM =
  GEN_ALL (RIGHT_BETA (REFL "(\f:*.f)x"));;

let four_tuple_lemma = TAC_PROOF
(([],"((aa:word9,bb:word9,cc:word9,dd:word9) =
      x1 => (a',b',c',d') |
      x2 => (a'',b'',c'',d'') |
            (a''',b''',c''',d''')) =
   (aa = x1 => a' | x2 => a'' | a''') /\
   (bb = x1 => b' | x2 => b'' | b''') /\
   (cc = x1 => c' | x2 => c'' | c''') /\
   (dd = x1 => d' | x2 => d'' | d''')"),
 COND_CASES_TAC THEN COND_CASES_TAC THEN art[PAIR_EQ]);;

%-------------------------------------------------------------%
% lemma to help unwind the state register definitions.       %
%-------------------------------------------------------------%
let state_reg_lemma = TAC_PROOF
(([],
 "STATE_REG Clocked reset load
         (next_mpc,next_s0,next_s1,next_s2,next_s3)
         (     mpc,     s0,     s1,     s2,     s3) =
 (!t.
   mpc(SUC t) =
   (Clocked t => (reset t => #000000000 | next_mpc t) | mpc t)) /\
 (!t.s0(SUC t) = (Clocked t =>(load t =>next_s0 t | s0 t) | s0 t)) /\
 (!t.s1(SUC t) = (Clocked t =>(load t =>next_s1 t | s1 t) | s1 t)) /\
 (!t.s2(SUC t) = (Clocked t =>(load t =>next_s2 t | s2 t) | s2 t)) /\
 (!t.s3(SUC t) = (Clocked t =>(load t =>next_s3 t | s3 t) | s3 t))"),
  prt[definition `rt_CU` `STATE_REG`;
      definition `rt_CU` `MPC9`;
      definition `rt_CU` `S_latch9`]
  THEN REFL_TAC);;

%-------------------------------------------------------------%
% The, entire Control unit unwound.                          %
%-------------------------------------------------------------%
let CU_unwound_lemma = prove_thm
(`CU_unwound_lemma`,
 "CU     Clocked
         reset button
         mpc s0 s1 s2 s3
         opcode
         atomflag bit30flag bit31flag zeroflag nilflag eqflag leqflag
         flag0 flag1
         ralu rmem rarg rbuf1 rbuf2 rcar rs re rc rd rmar rx1 rx2
         rfree rparent rroot ry1 ry2 rnum rnil rtrue rfalse rcons
         write_bit bidir
         warg wbuf1 wbuf2 wcar ws we wc wd wmar wx1 wx2
         wfree wparent wroot wy1 wy2
         dec add sub mul div rem
         setbit30 setbit31 resetbit31 replcar replcdr resetbit30 =
```

```
(!t.
  (mpc(SUC t) =
   (Clocked t =>
    (reset t =>
     #000000000 |
     ((((Test_field (ROM_fun (mpc t))) = #0001) \/
       ((Test_field (ROM_fun (mpc t))) = #0011) /\ bit30flag t \/
       ((Test_field (ROM_fun (mpc t))) = #0100) /\ bit31flag t \/
       ((Test_field (ROM_fun (mpc t))) = #0101) /\ eqflag t \/
       ((Test_field (ROM_fun (mpc t))) = #0110) /\ leqflag t \/
       ((Test_field (ROM_fun (mpc t))) = #0111) /\ nilflag t \/
       ((Test_field (ROM_fun (mpc t))) = #1000) /\ atomflag t \/
       ((Test_field (ROM_fun (mpc t))) = #1001) /\ zeroflag t \/
       ((Test_field (ROM_fun (mpc t))) = #1010) /\ button t \/
       ((Test_field (ROM_fun (mpc t))) = #1011)) =>
      (A_field (ROM_fun (mpc t))) |
      (((Test_field (ROM_fun (mpc t))) = #1100) =>
       s0 t |
       (((Test_field (ROM_fun (mpc t))) = #0010) =>
        opcode t |
        Inc9(mpc t))))) |
    mpc t)) /\
  (s0(SUC t) =
   (Clocked t =>
    ((((Test_field (ROM_fun (mpc t))) = #1011) \/
      ((Test_field (ROM_fun (mpc t))) = #1100)) =>
     (((Test_field (ROM_fun (mpc t))) = #1011) =>
      Inc9(mpc t) |
      (((Test_field (ROM_fun (mpc t))) = #1100) => s1 t | s0 t)) |
     s0 t) |
    s0 t)) /\
  (s1(SUC t) =
   (Clocked t =>
    ((((Test_field (ROM_fun (mpc t))) = #1011) \/
      ((Test_field (ROM_fun (mpc t))) = #1100)) =>
     (((Test_field (ROM_fun (mpc t))) = #1011) =>
      s0 t |
      (((Test_field (ROM_fun (mpc t))) = #1100) => s2 t | s1 t)) |
     s1 t) |
    s1 t)) /\
  (s2(SUC t) =
   (Clocked t =>
    ((((Test_field (ROM_fun (mpc t))) = #1011) \/
      ((Test_field (ROM_fun (mpc t))) = #1100)) =>
     (((Test_field (ROM_fun (mpc t))) = #1011) =>
      s1 t |
      (((Test_field (ROM_fun (mpc t))) = #1100) => s3 t | s2 t)) |
     s2 t) |
    s2 t)) /\
  (s3(SUC t) =
   (Clocked t =>
    ((((Test_field (ROM_fun (mpc t))) = #1011) \/
      ((Test_field (ROM_fun (mpc t))) = #1100)) =>
     (((Test_field (ROM_fun (mpc t))) = #1011) =>
```

```
    s2 t |
    (((Test_field (ROM_fun (mpc t))) = #1100) => #000000000 | s3 t)) |
   s3 t) |
  s3 t)) /\
(ralu t = ((Read_field (ROM_fun (mpc t))) = #00001)) /\
(rmem t = ((Read_field (ROM_fun (mpc t))) = #00010)) /\
(rarg t = ((Read_field (ROM_fun (mpc t))) = #00011)) /\
(rbuf1 t = ((Read_field (ROM_fun (mpc t))) = #00100)) /\
(rbuf2 t = ((Read_field (ROM_fun (mpc t))) = #00101)) /\
(rcar t = ((Read_field (ROM_fun (mpc t))) = #00110)) /\
(rs t = ((Read_field (ROM_fun (mpc t))) = #00111)) /\
(re t = ((Read_field (ROM_fun (mpc t))) = #01000)) /\
(rc t = ((Read_field (ROM_fun (mpc t))) = #01001)) /\
(rd t = ((Read_field (ROM_fun (mpc t))) = #01010)) /\
(rmar t = ((Read_field (ROM_fun (mpc t))) = #01011)) /\
(rx1 t = ((Read_field (ROM_fun (mpc t))) = #01100)) /\
(rx2 t = ((Read_field (ROM_fun (mpc t))) = #01101)) /\
(rfree t = ((Read_field (ROM_fun (mpc t))) = #01110)) /\
(rparent t = ((Read_field (ROM_fun (mpc t))) = #01111)) /\
(rroot t = ((Read_field (ROM_fun (mpc t))) = #10000)) /\
(ry1 t = ((Read_field (ROM_fun (mpc t))) = #10001)) /\
(ry2 t = ((Read_field (ROM_fun (mpc t))) = #10010)) /\
(rnum t = ((Read_field (ROM_fun (mpc t))) = #10011)) /\
(rnil t = ((Read_field (ROM_fun (mpc t))) = #10100)) /\
(rtrue t = ((Read_field (ROM_fun (mpc t))) = #10101)) /\
(rfalse t = ((Read_field (ROM_fun (mpc t))) = #10110)) /\
(rcons t = ((Read_field (ROM_fun (mpc t))) = #10111)) /\
(write_bit t = ~((Write_field(ROM_fun (mpc t))) = #00001)) /\
(bidir t = ~((Write_field(ROM_fun (mpc t))) = #00001)) /\
(warg t = ((Write_field(ROM_fun (mpc t))) = #00010)) /\
(wbuf1 t = ((Write_field(ROM_fun (mpc t))) = #00011)) /\
(wbuf2 t = ((Write_field(ROM_fun (mpc t))) = #00100)) /\
(wcar t = ((Write_field(ROM_fun (mpc t))) = #00101)) /\
(ws t = ((Write_field(ROM_fun (mpc t))) = #00110)) /\
(we t = ((Write_field(ROM_fun (mpc t))) = #00111)) /\
(wc t = ((Write_field(ROM_fun (mpc t))) = #01000)) /\
(wd t = ((Write_field(ROM_fun (mpc t))) = #01001)) /\
(wmar t = ((Write_field(ROM_fun (mpc t))) = #01010)) /\
(wx1 t = ((Write_field(ROM_fun (mpc t))) = #01011)) /\
(wx2 t = ((Write_field(ROM_fun (mpc t))) = #01100)) /\
(wfree t = ((Write_field(ROM_fun (mpc t))) = #01101)) /\
(wparent t = ((Write_field(ROM_fun (mpc t))) = #01110)) /\
(wroot t = ((Write_field(ROM_fun (mpc t))) = #01111)) /\
(wy1 t = ((Write_field(ROM_fun (mpc t))) = #10000)) /\
(wy2 t = ((Write_field(ROM_fun (mpc t))) = #10001)) /\
(dec t = ((Alu_field (ROM_fun (mpc t))) = #0001)) /\
(add t = ((Alu_field (ROM_fun (mpc t))) = #0010)) /\
(sub t = ((Alu_field (ROM_fun (mpc t))) = #0011)) /\
(mul t = ((Alu_field (ROM_fun (mpc t))) = #0100)) /\
(div t = ((Alu_field (ROM_fun (mpc t))) = #0101)) /\
(rem t = ((Alu_field (ROM_fun (mpc t))) = #0110)) /\
(setbit30 t = ((Alu_field (ROM_fun (mpc t))) = #0111)) /\
(setbit31 t = ((Alu_field (ROM_fun (mpc t))) = #1000)) /\
(resetbit31 t = ((Alu_field (ROM_fun (mpc t))) = #1001)) /\
```

```
    (replcar t = ((Alu_field (ROM_fun (mpc t))) = #1010)) /\
    (replcdr t = ((Alu_field (ROM_fun (mpc t))) = #1011)) /\
    (resetbit30 t = ((Alu_field (ROM_fun (mpc t))) = #1100)) /\
    (flag0 t = (mpc t = #000010110) \/ (mpc t = #000011000)) /\
    (flag1 t = (mpc t = #000101011) \/ (mpc t = #000011000))
  )",
  SUBST1_TAC
      (SPEC_ALL (definition `rt_CU` `CU`))      %  10.4s  4.7s   75 %
  THEN port [state_reg_lemma;
             definition `rt_CU` `CU_DECODE`;
             definition `rt_CU` `ROM_t`]        %  86.5s 51.3s 2147 %
  THEN prt [LET_DEF]                            % 273.8s 118.0s 12981 %
  THEN in1_conv_tac ETA_CONV                    %  66.0s 33.3s 3212 %
  THEN prt [ID_THM; four_tuple_lemma]           % ??? 265.3s 114.9s 12621 %
  THEN in_conv_tac BETA_CONV                    % ??? 192.4s 83.2s 9432 %
  THEN in_conv_tac (UNWINDF THENC PRUNEF)       % 443.5s 145.6s 10789 %
  THEN REFL_TAC                                 %  38.4s 79.8s 19 %
  );;

%-------------------------------------------------------------%
% A function to abstract out a subterm from a theorem        %
% conclusion, so the result is beta_convertable to the       %
% original.  This is useful when the subterm will explode    %
% in the course of simplification, which can be done in      %
% one place rather than multiple occurrences in the term,    %
% and the beta reduction accomplished only after the         %
% simplification is complete.                                %
%-------------------------------------------------------------%
let BETA_INV (t1,t2) thm =
  SUBS
    [SYM
     (BETA_CONV
      (mk_comb
       (mk_abs (t1, subst [t1,t2](concl thm)),
       t2)))]
  thm;;

let base_thm = ((SPEC "t:^mtime") o UNDISCH o fst o EQ_IMP_RULE)
         CU_unwound_lemma;;

%-----------------------------------------------------------------%
% a series of partition lemmas let us split the huge control      %
% unit into segments, for proofs of each possibiliity for         %
% sets of ouput signals, etc.                                     %
%                                                                 %
% state_base_lemma =                                              %
% . |- (\tests.                                                   %
%        (mpc(SUC t) =                                            %
%         (Clocked t =>                                           %
%          (reset t =>                                            %
%           #000000000 |                                          %
%           (((tests = #0001) \/                                  %
%             (tests = #0011) /\ bit30flag t \/                   %
%             (tests = #0100) /\ bit31flag t \/                   %
```

```
%            (tests = #0101) /\ eqflag t \/                          %
%            (tests = #0110) /\ leqflag t \/                         %
%            (tests = #0111) /\ nilflag t \/                         %
%            (tests = #1000) /\ atomflag t \/                        %
%            (tests = #1001) /\ zeroflag t \/                        %
%            (tests = #1010) /\ button t \/                          %
%            (tests = #1011)) =>                                     %
%           A_field(ROM_fun(mpc t)) |                                %
%           ((tests = #1100) =>                                      %
%            s0 t |                                                  %
%            ((tests = #0010) => opcode t | Inc9(mpc t))))) |        %
%         mpc t)) /\                                                 %
%        (s0(SUC t) =                                                %
%         (Clocked t =>                                              %
%          (((tests = #1011) \/ (tests = #1100)) =>                  %
%           ((tests = #1011) =>                                      %
%            Inc9(mpc t) |                                           %
%            ((tests = #1100) => s1 t | s0 t)) |                     %
%           s0 t) |                                                  %
%          s0 t)) /\                                                 %
%        (s1(SUC t) =                                                %
%         (Clocked t =>                                              %
%          (((tests = #1011) \/ (tests = #1100)) =>                  %
%           ((tests = #1011) => s0 t |                               %
%                               ((tests = #1100) => s2 t | s1 t)) |  %
%           s1 t) |                                                  %
%          s1 t)) /\                                                 %
%        (s2(SUC t) =                                                %
%         (Clocked t =>                                              %
%          (((tests = #1011) \/ (tests = #1100)) =>                  %
%           ((tests = #1011) => s1 t |                               %
%                               ((tests = #1100) => s3 t | s2 t)) |  %
%           s2 t) |                                                  %
%          s2 t)) /\                                                 %
%        (s3(SUC t) =                                                %
%         (Clocked t =>                                              %
%          (((tests = #1011) \/ (tests = #1100)) =>                  %
%           ((tests = #1011) =>                                      %
%            s2 t |                                                  %
%            ((tests = #1100) => #000000000 | s3 t)) |               %
%           s3 t) |                                                  %
%          s3 t)))                                                   %
%      (Test_field(ROM_fun(mpc t)))                                  %
% reads_base_lemma =                                                 %
% . |- (\reads.                                                      %
%        (ralu t = (reads = #00001)) /\                              %
%        (rmem t = (reads = #00010)) /\                              %
%        (rarg t = (reads = #00011)) /\                              %
%        (rbuf1 t = (reads = #00100)) /\                             %
%        (rbuf2 t = (reads = #00101)) /\                             %
%        (rcar t = (reads = #00110)) /\                              %
%        (rs t = (reads = #00111)) /\                                %
%        (re t = (reads = #01000)) /\                                %
%        (rc t = (reads = #01001)) /\                                %
```

```
%       (rd t = (reads = #01010)) /\                                    %
%       (rmar t = (reads = #01011)) /\                                  %
%       (rx1 t = (reads = #01100)) /\                                   %
%       (rx2 t = (reads = #01101)) /\                                   %
%       (rfree t = (reads = #01110)) /\                                 %
%       (rparent t = (reads = #01111)) /\                               %
%       (rroot t = (reads = #10000)) /\                                 %
%       (ry1 t = (reads = #10001)) /\                                   %
%       (ry2 t = (reads = #10010)) /\                                   %
%       (rnum t = (reads = #10011)) /\                                  %
%       (rnil t = (reads = #10100)) /\                                  %
%       (rtrue t = (reads = #10101)) /\                                 %
%       (rfalse t = (reads = #10110)) /\                                %
%       (rcons t = (reads = #10111)))                                   %
%     (Read_field(ROM_fun(mpc t)))                                      %
% writes_base_lemma =                                                   %
% . |- (\writes.                                                        %
%       (write_bit t = ~(writes = #00001)) /\                           %
%       (bidir t = ~(writes = #00001)) /\                               %
%       (warg t = (writes = #00010)) /\                                 %
%       (wbuf1 t = (writes = #00011)) /\                                %
%       (wbuf2 t = (writes = #00100)) /\                                %
%       (wcar t = (writes = #00101)) /\                                 %
%       (ws t = (writes = #00110)) /\                                   %
%       (we t = (writes = #00111)) /\                                   %
%       (wc t = (writes = #01000)) /\                                   %
%       (wd t = (writes = #01001)) /\                                   %
%       (wmar t = (writes = #01010)) /\                                 %
%       (wx1 t = (writes = #01011)) /\                                  %
%       (wx2 t = (writes = #01100)) /\                                  %
%       (wfree t = (writes = #01101)) /\                                %
%       (wparent t = (writes = #01110)) /\                              %
%       (wroot t = (writes = #01111)) /\                                %
%       (wy1 t = (writes = #10000)) /\                                  %
%       (wy2 t = (writes = #10001)))                                    %
%     (Write_field(ROM_fun(mpc t)))                                     %
% alus_base_lemma =                                                     %
% . |- (\alus.                                                          %
%       (dec t = (alus = #0001)) /\                                     %
%       (add t = (alus = #0010)) /\                                     %
%       (sub t = (alus = #0011)) /\                                     %
%       (mul t = (alus = #0100)) /\                                     %
%       (div t = (alus = #0101)) /\                                     %
%       (rem t = (alus = #0110)) /\                                     %
%       (setbit30 t = (alus = #0111)) /\                                %
%       (setbit31 t = (alus = #1000)) /\                                %
%       (resetbit31 t = (alus = #1001)) /\                              %
%       (replcar t = (alus = #1010)) /\                                 %
%       (replcdr t = (alus = #1011)) /\                                 %
%       (resetbit30 t = (alus = #1100)))                                %
%     (Alu_field(ROM_fun(mpc t)))                                       %
% outputs_base_lemma =                                                  %
% . |- (flag0 t = (mpc t = #000010110) \/ (mpc t = #000011000)) /\      %
%     (flag1 t = (mpc t = #000101011) \/ (mpc t = #000011000))          %
```

```
%-----------------------------------------------------------------------%
let [state_base_lemma; reads_base_lemma; writes_base_lemma;
      alus_base_lemma; outputs_base_lemma] =
  (\l. [BETA_INV ("tests:word4",
                  "Test_field(ROM_fun((mpc:num->word9)t))")
                 (LIST_CONJ (seg(0,4)l));
        BETA_INV ("reads:word5",
                  "Read_field(ROM_fun((mpc:num->word9)t))")
                 (LIST_CONJ (seg(5,27)l));
        BETA_INV ("writes:word5",
                  "Write_field(ROM_fun((mpc:num->word9)t))")
                 (LIST_CONJ (seg(28,45)l));
        BETA_INV ("alus:word4",
                  "Alu_field(ROM_fun((mpc:num->word9)t))")
                 (LIST_CONJ (seg(46,57)l));
        LIST_CONJ (seg(58,59)l)
       ]
  ) (CONJUNCTS base_thm);;

save_thm (`outputs_base_lemma`, outputs_base_lemma);;

%-----------------------------------------------------------------%
% derive the one_asserted'ness property of the alu outputs       %
% from their definition.  this is used later when the cu is       %
% composed with the dp, to drop the implication from the          %
% alu behavioural definition.                                     %
%-----------------------------------------------------------------%
let alucntl_one_asserted_lemma = prove_thm
(`alucntl_one_asserted_lemma`,
 "CU     Clocked
         reset button
         mpc s0 s1 s2 s3
         opcode
         atomflag bit30flag bit31flag zeroflag nilflag eqflag leqflag
         flag0 flag1
         ralu rmem rarg rbuf1 rbuf2 rcar rs re rc rd rmar rx1 rx2
         rfree rparent rroot ry1 ry2 rnum rnil rtrue rfalse rcons
         write_bit bidir
         warg wbuf1 wbuf2 wcar ws we wc wd wmar wx1 wx2
         wfree wparent wroot wy1 wy2
         dec add sub mul div rem
         setbit30 setbit31 resetbit31 replcar replcdr resetbit30
  ==>
  one_asserted_12 replcar replcdr sub add dec mul div rem
                  setbit30 setbit31 resetbit30 resetbit31
 ",
 DISCH_THEN                                    % 137.0s 87.3s 4965 %
 ((\thl. port[one_asserted_12]
   THEN GEN_TAC
   THEN (REPEAT CONJ_TAC)
   THEN port thl) o
   (seg (46,57)) o
   CONJUNCTS o
   (SPEC "t:^mtime") o
```

```
   (SUBST [CU_unwound_lemma,"xxx:bool"]"xxx:bool"))
 THEN DISCH_THEN
 (\th. SUBST1_TAC th
  THEN rt[theorem `CU_wordn_proofs` ((`word_`^(implode o tl o explode o
                                          fst o dest_const o snd o
                                          dest_eq o concl)th))])
 );;                      % total for proof : 256.7s 239.8s 10044 %

let word4_EQ_CONV = wordn_EQ_CONV (theorem `cu_types` `Word4_11`);;
let word5_EQ_CONV = wordn_EQ_CONV (theorem `cu_types` `Word5_11`);;
let word9_EQ_CONV = wordn_EQ_CONV (theorem `cu_types` `Word9_11`);;

%-------------------------------------------------------------%
% we need lemmas for each used mpc value, giving the outputs %
% of each signal.  this should be generated somewhat         %
% automatically if feasible....  this first effort will      %
% ensure the form of the lemmas is reasonable.               %
%-------------------------------------------------------------%

% ============================================================ %
% convert a num to a boolean list (1's and 0's).               %
%                                                              %
% functions:                                                   %
%   mk_bit_list (num,size) res                                 %
%       returns a list of strings of 1's and 0's,              %
%       of length size.                                        %
%   example : #mk_bit_list (3,4)[];;                           %
%             [`0`; `0`; `1`; `1`] : string list               %
% ============================================================ %
let rem (x,y) = x - y * (x / y);;

letrec mk_bit_list (num, size) res =
   (size = 0) => res |
                 (mk_bit_list ((num/2), (size-1))
                   (((string_of_int o rem) (num,2)) . res));;

% ============================================================ %
% two functions for creating constants.                        %
% these 2 functions return the appropriate wordn object, from  %
% a pair or list of pairs.                                     %
% ============================================================ %
let mk_word9_from_num n =
        mk_const
           (implode (`#` . (mk_bit_list (n,9) ([]:(string)list))),
           ":word9");;

letrec int_of_bin l v =
  (l = []) => v |
              (int_of_bin (tl l) (v + v + ((hd l = `0`)=> 0 | 1)));;
% ============================================================ %
% here is the stuff necessary for the major proof function     %
% below.  including:                                           %
%   - definitions of field selector functions from cu_types    %
%   - a base theorem for the forward proofs                    %
```

```
%   - a conversion for expanding word27 constants only         %
%   - function "clear" to strip redundancies from a list       %
% ========================================================== %
map (load_definition `cu_types`)
  [`A_field`; `Write_field`; `Read_field`; `Alu_field`; `Test_field`];;


let generic_prove_fcn (str,ty) thm =
  let tm = (snd o dest_comb o concl) thm
  and stem = (implode o tl o explode) str
  in
  let assum = mk_eq(tm,mk_const(str,ty))
  in
  let simp = theorem `CU_wordn_proofs` (`word_`^stem)
  in
  let th1 = SUBS [ASSUME assum] thm
  in
  save_thm
   ((((fst o dest_const o fst o dest_comb) tm)^`_`^stem),
    (DISCH_ALL o (prr [AND_CLAUSES;OR_CLAUSES;
                       COND_CLAUSES;NOT_CLAUSES])
               o (SUBS (CONJUNCTS simp)))
               (SUBS [BETA_CONV (concl th1)] th1));;

letrec proof_series n ty thm =
  (n < 0) =>
     truth |
     ((let len = (int_of_string o hd o tl o tl o tl o tl o
                  explode o fst o dest_type)ty
       in
       generic_prove_fcn
        (implode(`#`.(mk_bit_list(n,len)([]:(string)list))),ty) thm) ;
       proof_series (n-1) ty thm);;

% ========================================================== %
% prove the whole lot of therorems about the various         %
% possible values for things.                                %
% ========================================================== %
proof_series 12 ":word4" alus_base_lemma;;
proof_series 12 ":word4"
  (rr[SUBS [SPEC "Clocked:num->bool"
                 (definition `constraints` `clock_constraint`)]
           (ASSUME "clock_constraint Clocked")]
     state_base_lemma);;
proof_series 23 ":word5" reads_base_lemma;;
proof_series 17 ":word5" writes_base_lemma;;

close_theory ();;
print_theory `-`;;
```

4.2.4 prove_CU_lemmas

```
%                                                                      %
% FILE: prove_CU_lemmas.ml                                             %
%                                                                      %
% DESCRIPTION: Provides a proof function that returns a theorem        %
%              about the value of the outputs and the next state       %
%              of the control unit for any valid mpc value.            %
%                                                                      %
%               top level function:                                    %
%                 prove_CU_lemma :(:word9)term -> theorem              %
%                                                                      %
% USES FILES:  CU_proofs.th, wordn library                             %
%                                                                      %
% Brian Graham 89.09.18                                                %
%                                                                      %
% Modifications:                                                       %
%                                                                      %
%======================================================================%

% ============================================================ %
% In order to use this file, one must be in a theory with      %
% the following conditions holding:                            %
%  * the wordn library must be loaded                          %
%  * CU_proofs must be an ancestor theory                      %
% ============================================================ %

load_theorem `CU_proofs` `outputs_base_lemma`;;

map (load_definition `cu_types`)
  [`A_field`; `Write_field`; `Read_field`; `Alu_field`; `Test_field`];;

letrec int_of_bin l v =
  (l = []) => v |
              (int_of_bin (tl l) (v + v + ((hd l = `0`)=> 0 | 1)));;

let word9_EQ_CONV = wordn_EQ_CONV (theorem `cu_types` `Word9_11`);;

% ============================================================ %
% The main proof function.  This is designed to produce any    %
% of the 400 lemmas required, evaluating all outputs of the    %
%  control unit for a given mpc value.                         %
%                                                              %
% It is not fast, and may yet require optimization.  Recorded  %
% timing (for arg = "#000111000") is :                         %
% Run time: 38.8s                                              %
% Garbage collection time: 48.4s                               %
% Intermediate theorems generated: 1153                        %
% ============================================================ %
let prove_CU_lemma addr =
  let assum = "(mpc:num->word9) t = ^addr"
  and add_tok = tok_of_int(int_of_bin ((tl o explode o fst o dest_const) addr) 0)
  in                                % 0.1s 0.0s %
  let ROM_lemma = SUBS [SYM (ASSUME assum)]
```

```
                    (theorem `microcode` (`ROM_fun`^add_tok))
in                                  % 0.1s 0.0s 3 %
let char_list = (explode o fst o dest_const o snd o dest_eq o concl)
                ROM_lemma
in                                  % 0.0s 0.0s %
let ROM_lemma' = (CONV_RULE o DEPTH_CONV o CHANGED_CONV)
                wordn_CONV ROM_lemma
in                                   % 0.6s 0.0s 30 %
let Test_field_lemma =         % 7.6s 15.2s 254 %
  (let Test_val = implode(seg(10,13)char_list)
  in
  UNDISCH_ALL
  (MP (SUBS[((theorem `Inc9_proofs` (`Inc9_lem_`^add_tok)));
            SUBS [(SYM o wordn_CONV o mk_const)
                  (implode(seg(0,9)char_list),":word9")]
                 (PURE_ONCE_REWRITE_RULE
                     [A_field] (AP_TERM "A_field" ROM_lemma'))
           ]
           (theorem `CU_proofs` (`Test_field_`^Test_val)))
      (SUBS[(SYM o wordn_CONV o mk_const)(`#`^Test_val,":word4")]
           (PURE_ONCE_REWRITE_RULE
                [Test_field] (AP_TERM "Test_field" ROM_lemma')))))
and Read_field_lemma =         % 5.2s 0.0s 127 %
  (let Read_val = implode(seg(18,22)char_list)
  in
  UNDISCH_ALL
  (MP (theorem `CU_proofs` (`Read_field_`^Read_val))
      (SUBS[(SYM o wordn_CONV o mk_const)(`#`^Read_val,":word5")]
           (PURE_ONCE_REWRITE_RULE[Read_field](AP_TERM "Read_field" ROM_lemma')))))
and Alu_field_lemma =                % 4.5s 0.0s 127 %
  (let Alu_val = implode(seg(14,17)char_list)
  in
  UNDISCH_ALL
  (MP (theorem `CU_proofs` (`Alu_field_`^Alu_val))
      (SUBS[(SYM o wordn_CONV o mk_const)(`#`^Alu_val,":word4")]
           (PURE_ONCE_REWRITE_RULE[Alu_field](AP_TERM "Alu_field" ROM_lemma')))))
and Write_field_lemma =         % 4.9s 0.0s 127 %
  (let Write_val = implode(seg(23,27)char_list)
  in
  UNDISCH_ALL
  (MP (theorem `CU_proofs` (`Write_field_`^Write_val))
      (SUBS[(SYM o wordn_CONV o mk_const)(`#`^Write_val,":word5")]
           (PURE_ONCE_REWRITE_RULE[Write_field](AP_TERM "Write_field" ROM_lemma')))))
and flags_lemma =                % 10.5s 15.8s 316 %
    (SUBS (map (\tm. word9_EQ_CONV
                        (mk_eq (addr,tm)))
               ["#000010110";"#000011000";"#000101011"])
          (theorem `CU_proofs` `outputs_base_lemma`))
in                                   % 4.0s 0.0s 169 %
(LIST_CONJ o flat o (map CONJUNCTS))
    [Test_field_lemma;
     Alu_field_lemma;
     Read_field_lemma;
     Write_field_lemma;
```

```
      flags_lemma]
;;
```

4.2.5 DP_proofs

```
%                                                                      %
% FILE: DP_proofs.ml                                                   %
%                                                                      %
% DESCRIPTION: This proves some properties of the register-            %
%              transfer definition of the datapath.                    %
%                                                                      %
% USES FILES:  rt_DP.th,                                               %
%              wordn library                                           %
%                                                                      %
% Brian Graham 89.10.04                                                %
%                                                                      %
% Modifications:                                                       %
%                                                                      %
%======================================================================%

new_theory `DP_proofs`;;

loadf `load_wordn`;;

new_parent `rt_DP`;;

let ftime = ":num"
and mtime = ":num"
and ctime = ":num";;

let fsig = ":^ftime->bool"
and msig = ":^mtime->bool"
and csig = ":^ctime->bool";;

let w2_fvec = ":^ftime->word2"
and w2_mvec = ":^mtime->word2"
and w2_cvec = ":^ctime->word2";;

let w14_fvec = ":^ftime->word14"
and w14_mvec = ":^mtime->word14"
and w14_cvec = ":^ctime->word14";;

let w28_fvec = ":^ftime->word28"
and w28_mvec = ":^mtime->word28"
and w28_cvec = ":^ctime->word28";;

let w32_fvec = ":^ftime->word32"
and w32_mvec = ":^mtime->word32"
and w32_cvec = ":^ctime->word32";;

load_all `rt_DP`;;
load_all `dp_types`;;
load_library `unwind`;;

let ID_THM =
  GEN_ALL (RIGHT_BETA (REFL "(\f:*.f)x"));;
```

```
% ============================================================== %
% First, unwind the datapath, and see how simplified an expression  %
% can be obtained.                                                  %
% ============================================================== %

set_goal
(["!t:^mtime. Clocked t";
  "one_asserted_12
   replcar
   replcdr
   sub
   add
   dec
   mul
   div
   rem
   setbit30
   setbit31
   resetbit30
   resetbit31"],
 "DP bus bus_in Clocked
     rmem
     mar      wmar      rmar      rnum rnil rtrue rfalse
     s'       ws        rs
     e'       we        re
     c'       wc        rc
     d'       wd        rd
     free     wfree     rfree
     parent   wparent   rparent
     root     wroot     rroot
     y1       wy1       ry1
     x1       wx1       rx1
     x2       wx2       rx2
     y2       wy2       ry2       rcons
     car      wcar      rcar
     atomflag bit30flag bit31flag zeroflag nilflag eqflag leqflag
     arg      warg      rarg
     buf1     wbuf1     rbuf1
     buf2     wbuf2     rbuf2
     replcar  replcdr   sub add dec mul div rem
     setbit30 setbit31  resetbit30 resetbit31
     ralu ==>
   ((!t. rmem t   ==> (bus t           = bus_in t))        /\
    (!t. rmar t   ==> (cdr_bits(bus t) = mar t))          /\
    (!t. rnum t   ==> (cdr_bits(bus t) = #11111111111111)) /\
    (!t. rnil t   ==> (cdr_bits(bus t) = #00000000000000)) /\
    (!t. rtrue t  ==> (cdr_bits(bus t) = #00000000000001)) /\
    (!t. rfalse t ==> (cdr_bits(bus t) = #00000000000010)) /\
    (!t. rs t     ==> (cdr_bits(bus t) = s' t))           /\
    (!t. re t     ==> (cdr_bits(bus t) = e' t))           /\
    (!t. rc t     ==> (cdr_bits(bus t) = c' t))           /\
    (!t. rd t     ==> (cdr_bits(bus t) = d' t))           /\
    (!t. rfree t  ==> (cdr_bits(bus t) = free t))         /\
    (!t. rx1 t    ==> (cdr_bits(bus t) = x1 t))           /\
```

```
   (!t. rx2 t    ==> (cdr_bits(bus t) = x2 t))           /\

   ((!t. rcons t ==> (garbage_bits(bus t)  = #00))       /\
    (!t. rcons t ==> (rec_type_bits(bus t) = RT_CONS))   /\
    (!t. rcons t ==> (car_bits(bus t)      = x1 t))      /\
    (!t. rcons t ==> (cdr_bits(bus t)      = x2 t)))     /\

   (!t. rcar t  ==> (cdr_bits(bus t) = car  t))         /\
   (!t. rarg t  ==> (bus t           = arg  t))         /\
   (!t. rbuf1 t ==> (bus t           = buf1 t))         /\
   (!t. rbuf2 t ==> (bus t           = buf2 t))         /\

  (?alu.
   (!t. ralu t  ==> (bus t           = alu  t))         /\

   (!t. sub t ==> (alu t = SUB28(atom_bits(arg t))
                           (atom_bits(bus t))))       /\
   (!t. add t ==> (alu t = ADD28(atom_bits(arg t))
                           (atom_bits(bus t))))       /\
   (!t. dec t ==> (alu t = DEC28(atom_bits(bus t))))  /\
   (!t. mul t ==> (alu t = DEC28(atom_bits(bus t))))  /\
   (!t. div t ==> (alu t = DEC28(atom_bits(bus t))))  /\
   (!t. rem t ==> (alu t = DEC28(atom_bits(bus t))))  /\

   (!t. buf1(t + 1) = (wbuf1(t + 1) => alu(t + 1) | buf1 t)) /\
   (!t. buf2(t + 1) = (wbuf2(t + 1) => alu(t + 1) | buf2 t))) /\

   (!t. mar (t + 1) = (wmar(t + 1)  => cdr_bits(bus(t + 1)) | mar t)) /\
   (!t. s'  (t + 1) = (ws(t + 1)    => cdr_bits(bus(t + 1)) | s'  t)) /\
   (!t. e'  (t + 1) = (we(t + 1)    => cdr_bits(bus(t + 1)) | e'  t)) /\
   (!t. c'  (t + 1) = (wc(t + 1)    => cdr_bits(bus(t + 1)) | c'  t)) /\
   (!t. d'  (t + 1) = (wd(t + 1)    => cdr_bits(bus(t + 1)) | d'  t)) /\
   (!t. free(t + 1) = (wfree(t + 1) => cdr_bits(bus(t + 1)) | free t))/\
   (!t. x1  (t + 1) = (wx1(t + 1)   => cdr_bits(bus(t + 1)) | x1  t)) /\
   (!t. x2  (t + 1) = (wx2(t + 1)   => cdr_bits(bus(t + 1)) | x2  t)) /\
   (!t. car (t + 1) = (wcar(t + 1)  => car_bits(bus(t + 1)) | car t)) /\
   (!t. arg (t + 1) = (warg(t + 1)  => bus(t + 1) | arg  t)) /\

   (!t. atomflag  t = is_atom(bus t))                    /\
   (!t. bit30flag t = field_bit(bus t))                  /\
   (!t. bit31flag t = mark_bit(bus t))                   /\
   (!t. zeroflag  t = (atom_bits(bus t) = ZERO28))       /\
   (!t. nilflag   t = (cdr_bits(bus t) = #00000000000000)) /\
   (!t. eqflag    t = (bus t = arg t))                   /\
   (!t. leqflag   t = LEQ_prim(arg t)(bus t)))
");;


expand
 (SUBST1_TAC (SPEC_ALL DP)
  THEN port[ READ_MEM; MAR; NUM; Nil; TRUE; FALSE;
             s; e; c; d; FREE; PARENT; ROOT; Y1; X1;
             X2; Y2; ARG; ALU; BUF1; BUF2 ]
  THEN prt[ register14; reg14; register32; reg32;
```

```
          busgate32; busgate14; busgate2;
          ZEROS14; ZERO28;
          NUM_addr; NIL_addr; T_addr; F_addr;
          Cons; Car; FLAGSUNIT ]
THEN port[ LET_DEF ]
THEN in1_conv_tac ETA_CONV
THEN prt[ID_THM]
THEN re_conv_tac BETA_CONV
THEN poart[]
THEN prt[ COND_CLAUSES; IMP_CLAUSES ]
THEN STRIP_TAC
THEN part[]
THEN rt[]
THEN EXISTS_TAC
THEN poart[]
THEN rt[]);;
```

4.3 Data and Temporal Abstractions

The RTL and Top level descriptions differ on two fundamental points: the memories are represented quite differently, and the granularity of time is distinct. In order to relate these, abstraction functions are defined.

4.3.1 mem_abs

```
% SECD verification                                                    %
%                                                                     %
% FILE:        mem_abs.ml                                             %
%                                                                     %
% DESCRIPTION: Defines an abstraction between the "real" memory       %
%              definition, and the abstract memories.                 %
%                                                                     %
% USES FILES:  integer library, dp_types.th, abstract_mem_type.th     %
%                                                                     %
% Brian Graham 89.08.01                                              %
%                                                                     %
% Modifications:                                                     %
%                                                                     %
%=====================================================================%
new_theory `mem_abs`;;

loadt `load_wordn`;;

load_library `integer`;;

new_parent `dp_types`;;
new_parent `abstract_mem_type`;;

let Mem_Range_Abs = new_definition
 (`Mem_Range_Abs`,
  "(Mem_Range_Abs:word32->((bool#bool)#((word14#word14)+atom))) w
    =
    ((mark_bit w),(field_bit w)),
    ((is_symbol w) => INR (Symb (Val  (Bits28 (atom_bits w))))     |
     (is_number w) => INR (Int  (iVal (Bits28 (atom_bits w))))     |
     (is_cons   w) => INL ((car_bits w), (cdr_bits w))             |
                      (@x.F))"               % UNUSED RECORD CASE %
 );;

let mem_abs = new_definition
  (`mem_abs`,
   "mem_abs (M:word14->word32) = ABS_mfsexp_mem (Mem_Range_Abs o M)"
  );;

close_theory ();;
print_theory `-`;;
```

4.3.2 when

Note: this file was written by I.S.Dhingra based on work by Tom Melham.

```
%-----------------------------------------------------------------------
| FILE          : when.ml
|
| DESCRIPTION   : Defines the predicates 'Next', 'Inf', 'IsTimeOf'
|                 and 'TimeOf' and derives several major theorems
|                 which provide a basis for temporal abstraction.
|
|                 These predicates and theorems are taken from
|                 T.Melham's paper, "Abstraction Mechanisms for
|                 Hardware Verification", Hardware Verification
|                 Workshop, University of Calgary, January 1987.
|
-----------------------------------------------------------------------%

new_theory `when`;;

let Next = new_definition
  (`Next`,
   "Next t1 t2 f  =  (t1<t2)  /\
                     ( f t2)  /\
                  !t. (t1<t )  /\  (t<t2)  ==>  ~f t"
  );;

let IsTimeOf = new_prim_rec_definition
  (`IsTimeOf`,
  "(IsTimeOf      0  f t  =  f t  /\  !t'.  (t'<t) ==> ~f t') /\
   (IsTimeOf (SUC n) f t  =  ?t'.  IsTimeOf n f t' /\
                                   Next t' t f              )"
  );;

let TimeOf = new_definition
  (`TimeOf`,
   "TimeOf f n  =  @t. IsTimeOf n f t"
  );;

let when = new_infix_definition
  (`when`,
   "when (s:num->*) (p:num->bool)  =  \n. s (TimeOf p n)"
  );;

let Inf = new_definition
  (`Inf`,
   "Inf f =  !t. ?t'.  (t<t') /\ (f t')"
  );;

%-----------------------------------------------------------------------
| Define "LEAST P" to represent that P has a smallest element.
-----------------------------------------------------------------------%
let LEAST = new_definition
  (`LEAST`,
```

```
   "LEAST P  =  ?x. P x  /\  (!y. y<x ==> ~P y)"
 );;

close_theory();;

%-----------------------------------------------------------------------
|   wop = |- !P.  (?n. P n)  ==>  LEAST P
-----------------------------------------------------------------------%
let wop = prove_thm
  ('wop',
   "!P. (?n. P n)  ==>  LEAST P",
   REWRITE_TAC [WOP; LEAST]
  );;

%-----------------------------------------------------------------------
|   Inf_EXISTS = |- !f.  Inf f  ==>  ?n.  f n
-----------------------------------------------------------------------%
let Inf_EXISTS = prove_thm
  ('Inf_EXISTS',
   "!f.  Inf f  ==>  ?n.  f n",
   PURE_REWRITE_TAC [Inf]
   THEN REPEAT STRIP_TAC
   THEN FIRST_ASSUM (STRIP_ASSUME_TAC  o (SPEC "t:num"))
   THEN EXISTS_TAC "t':num"
   THEN FIRST_ASSUM ACCEPT_TAC
  );;

%-----------------------------------------------------------------------
|   Inf_LEAST = |- !f.  Inf f  ==>  LEAST f
-----------------------------------------------------------------------%
let Inf_LEAST = prove_thm
  ('Inf_LEAST',
   "!f.  Inf f  ==>  LEAST f",
   REPEAT STRIP_TAC
   THEN IMP_RES_TAC Inf_EXISTS
   THEN IMP_RES_TAC wop
  );;

%-----------------------------------------------------------------------
|   Inf_Next = |- !f. Inf f ==> !t. f t ==> ?t'. Next t t' f
-----------------------------------------------------------------------%
let Inf_Next = prove_thm
  ('Inf_Next',
   "!f. Inf f ==> !t. f t ==> ?t'. Next t t' f",
   PURE_REWRITE_TAC [Inf; Next]
   THEN REPEAT (X_GEN_TAC "v:num" ORELSE STRIP_TAC)
   THEN RULE_ASSUM_TAC (\th. SPEC "v:num" th ? th)
   THEN IMP_RES_THEN (X_CHOOSE_THEN "n:num" STRIP_ASSUME_TAC) wop'
   THEN EXISTS_TAC "n:num"
   THEN ASM_REWRITE_TAC []
   THEN REPEAT STRIP_TAC
   THEN RES_THEN (STRIP_ASSUME_TAC o (REWRITE_RULE[DE_MORGAN_THM]))
   THEN RES_TAC
  )
```

```
where wop' =
 CONV_RULE (DEPTH_CONV BETA_CONV) (SPEC (( mk_abs
                                          o dest_exists
                                          o snd
                                          o dest_forall
                                          o rhs
                                          o concl
                                          o SPEC_ALL
                                          ) Inf)
                                         WOP
                                   );;

%----------------------------------------------------------------------
|   Next_ADD1 = |- !f t.  f (t+1)  ==>  Next t (t+1) f
----------------------------------------------------------------------%
let Next_ADD1 = prove_thm
  (`Next_ADD1`,
   "!f t.  f (t+1)  ==>  Next t (t+1) f",
   REWRITE_TAC [ Next
               ; SYM (SPEC_ALL ADD1)
               ; LESS_SUC_REFL
               ]
   THEN REPEAT STRIP_TAC
   THENL [ FIRST_ASSUM ACCEPT_TAC
         ; IMP_RES_THEN
             (STRIP_ASSUME_TAC o (CONV_RULE (ONCE_DEPTH_CONV SYM_CONV)))
             LESS_NOT_EQ
          THEN IMP_RES_TAC LESS_SUC_IMP
          THEN IMP_RES_TAC LESS_ANTISYM
         ]
  );;

%----------------------------------------------------------------------
|   Next_INCREAST = |- !f t1 t2.   ~f(t1+1)       ==>
|                                Next (t1+1) t2 f  ==>  Next t1 t2 f
----------------------------------------------------------------------%
let Next_INCREASE = prove_thm
  (`Next_INCREASE`,
   "!f t1 t2.   ~f(t1+1)       ==>
             Next (t1+1) t2 f  ==>  Next t1 t2 f",
   PURE_REWRITE_TAC [Next; SYM (SPEC_ALL ADD1)]
   THEN REPEAT STRIP_TAC
   THENL [ IMP_RES_TAC SUC_LESS
         ; FIRST_ASSUM ACCEPT_TAC
         ; MATCH_UNDISCH_TAC "~^(genvar":bool")"
           THEN IMP_RES_TAC (PURE_REWRITE_RULE [LESS_OR_EQ] LESS_SUC_EQ)
           THEN RES_TAC
           THEN ASM_REWRITE_TAC []
        ]
  );;

%----------------------------------------------------------------------
|   Next_IDENTITY = |- !t1 t2 f.   Next t1 t2 f  ==>
|                           !t3.   Next t1 t3 f  ==>  (t2 = t3)
```

```
------------------------------------------------------------------------%
let Next_IDENTITY = prove_thm
  (`Next_IDENTITY`,
   "!t1 t2 f.   Next t1 t2 f  ==>
          !t3.  Next t1 t3 f  ==>  (t2 = t3)",
   PURE_REWRITE_TAC [Next]
   THEN REPEAT STRIP_TAC
   THEN PURE_ONCE_REWRITE_TAC
          [(SYM o SPEC_ALL o hd o CONJUNCTS) NOT_CLAUSES]
   THEN DISCH_TAC
   THEN STRIP_ASSUME_TAC
          (SPECL ["t2:num"; "t3:num"]
                 (REWRITE_RULE [DE_MORGAN_THM] LESS_ANTISYM))
   THENL [ ALL_TAC
         ; RULE_ASSUM_TAC (CONV_RULE (ONCE_DEPTH_CONV SYM_CONV))
         ]
   THEN IMP_RES_TAC LESS_CASES_IMP
   THEN RES_TAC
  );;

%------------------------------------------------------------------------
|   IsTimeOf_TRUE = |- !n f t.  IsTimeOf n f t ==> f t
------------------------------------------------------------------------%
let IsTimeOf_TRUE = prove_thm
  (`IsTimeOf_TRUE`,
   "!n f t.  IsTimeOf n f t ==> f t",
   INDUCT_TAC
   THEN REWRITE_TAC [IsTimeOf; Next]
   THEN REPEAT STRIP_TAC
  );;

%------------------------------------------------------------------------
|   IsTimeOf_EXISTS = |- !f. Inf f ==> !n. ?t. IsTimeOf n f t
------------------------------------------------------------------------%
let IsTimeOf_EXISTS = prove_thm
  (`IsTimeOf_EXISTS`,
   "!f. Inf f ==> !n. ?t. IsTimeOf n f t",
   GEN_TAC
   THEN DISCH_TAC
   THEN INDUCT_TAC
   THENL [ IMP_RES_TAC Inf_EXISTS
           THEN IMP_RES_THEN
                  (ASSUME_TAC o (PURE_REWRITE_RULE [LEAST]))
                  Inf_LEAST
           THEN ASM_REWRITE_TAC [IsTimeOf]
         ; FIRST_ASSUM STRIP_ASSUME_TAC
           THEN IMP_RES_TAC IsTimeOf_TRUE
           THEN IMP_RES_TAC Inf_Next
           THEN FIRST_ASSUM STRIP_ASSUME_TAC
           THEN REWRITE_TAC [IsTimeOf]
           THEN EXISTS_TAC "t':num"
           THEN EXISTS_TAC "t:num"
           THEN ASM_REWRITE_TAC []
         ]
```

```
  );;

%----------------------------------------------------------------------
|   TimeOf_DEFINED = |- !f. Inf f ==> (!n. IsTimeOf n f (TimeOf f n))
----------------------------------------------------------------------%
let TimeOf_DEFINED = save_thm
  (`TimeOf_DEFINED`, ( GEN_ALL
                     o DISCH_ALL
                     o GEN_ALL
                     o (REWRITE_RULE [SYM(SPEC_ALL TimeOf)])
                     o CONV_RULE (DEPTH_CONV BETA_CONV)
                     o (REWRITE_RULE [EXISTS_DEF])
                     o SPEC_ALL
                     o UNDISCH_ALL
                     o SPEC_ALL
                     ) IsTimeOf_EXISTS
  );;

%----------------------------------------------------------------------
|   TimeOf_TRUE = |- !f. Inf f ==> (!n. f (TimeOf f n))
----------------------------------------------------------------------%
let TimeOf_TRUE = save_thm
  (`TimeOf_TRUE`, ( GEN_ALL
                  o DISCH_ALL
                  o GEN_ALL
                  o (MATCH_MP IsTimeOf_TRUE)
                  o SPEC_ALL
                  o UNDISCH_ALL
                  o SPEC_ALL
                  ) TimeOf_DEFINED
  );;

%----------------------------------------------------------------------
|   IsTimeOf_IDENTITY =
|   |- !n f t1 t2. IsTimeOf n f t1 /\ IsTimeOf n f t2 ==> (t1 = t2)
----------------------------------------------------------------------%
let IsTimeOf_IDENTITY = prove_thm
  (`IsTimeOf_IDENTITY`,
   "!n f t1 t2. IsTimeOf n f t1 /\ IsTimeOf n f t2 ==> (t1 = t2)",
   INDUCT_TAC
   THEN PURE_REWRITE_TAC [IsTimeOf; Next]
   THEN X_GEN_TAC "f:num->bool"
   THEN X_GEN_TAC "t1:num"
   THEN X_GEN_TAC "t2:num"
   THEN REPEAT STRIP_TAC
   THENL [ ALL_TAC
         ; RES_THEN
             (\th. EVERY_ASSUM
                     (STRIP_ASSUME_TAC o (\thm. SUBS [th] thm? thm)))
         ]
   THEN STRIP_ASSUME_TAC
          (SPECL ["t2:num"; "t1:num"]
                 (REWRITE_RULE [LESS_OR_EQ] LESS_CASES))
   THEN RES_TAC
```

```
  );;

%-----------------------------------------------------------------------
|   TimeOf_INCREASING =
|   |- !f. Inf f  ==>  !n. (TimeOf f n) < (TimeOf f (n+1))
-----------------------------------------------------------------------%
let TimeOf_INCREASING = prove_thm
  (`TimeOf_INCREASING`,
   "!f. Inf f ==> (!n. (TimeOf f n) < (TimeOf f(n+1)))",
   GEN_TAC
   THEN DISCH_TAC
   THEN X_GEN_TAC "n:num"
   THEN IMP_RES_TAC Inf_Next
   THEN IMP_RES_THEN (STRIP_ASSUME_TAC o (SPEC "n:num")) TimeOf_DEFINED
   THEN IMP_RES_THEN
         ( CHOOSE_THEN ( STRIP_ASSUME_TAC
                       o (\thl. CONJ (el 1 thl) (el 2 thl))
                       o CONJUNCTS
                       )
         o (REWRITE_RULE [IsTimeOf; Next])
         o (SPEC "SUC n")
         ) TimeOf_DEFINED
   THEN MATCH_UNDISCH_TAC "x<y"
   THEN IMP_RES_TAC IsTimeOf_TRUE
   THEN IMP_RES_THEN
         (\th. ONCE_REWRITE_TAC[ADD1; th]) IsTimeOf_IDENTITY
  );;

%-----------------------------------------------------------------------
|   TimeOf_INTERVAL =
|   |- !f. Inf f ==>
|      !n t. (TimeOf f n)<t  /\  t<(TimeOf f (n+1)) ==> ~f t
-----------------------------------------------------------------------%
let TimeOf_INTERVAL = prove_thm
  (`TimeOf_INTERVAL`,
   "!f. Inf f ==>
    !n t. (TimeOf f n)<t  /\  t<(TimeOf f (n+1)) ==> ~f t",
   GEN_TAC
   THEN DISCH_TAC
   THEN X_GEN_TAC "n:num"
   THEN X_GEN_TAC "t:num"
   THEN IMP_RES_TAC Inf_Next
   THEN IMP_RES_THEN (STRIP_ASSUME_TAC o (SPEC "n:num")) TimeOf_DEFINED
   THEN IMP_RES_THEN
         ( CHOOSE_THEN ( STRIP_ASSUME_TAC
                       o (\thl.  CONJ (el 1 thl) (SPEC "t:num" (el 4 thl)))
                       o CONJUNCTS
                       )
         o (REWRITE_RULE [IsTimeOf; Next])
         o (SPEC "SUC n")
         ) TimeOf_DEFINED
   THEN FIRST_ASSUM (UNDISCH_TAC o concl)
   THEN IMP_RES_TAC IsTimeOf_TRUE
   THEN IMP_RES_THEN  (\th. ONCE_REWRITE_TAC[ADD1; th])  IsTimeOf_IDENTITY
```

```
  );;

%----------------------------------------------------------------------
|   TimeOf_Next = |- !f. Inf f ==> !n. Next (TimeOf f n) (TimeOf f (n+1)) f
----------------------------------------------------------------------%
let TimeOf_Next = prove_thm
  (`TimeOf_Next`,
   "!f. Inf f ==> !n. Next (TimeOf f n) (TimeOf f (n+1)) f",
   PURE_REWRITE_TAC [Next]
   THEN REPEAT STRIP_TAC
   THENL [ IMP_RES_THEN (\th. REWRITE_TAC [th]) TimeOf_INCREASING
         ; IMP_RES_THEN (\th. REWRITE_TAC [th]) TimeOf_TRUE
         ; IMP_RES_TAC TimeOf_INTERVAL THEN RES_TAC
         ]
  );;

print_theory `when`;;
%<----------------------------------------------------------------------
The Theory when
Parents --  HOL     wop
Constants --
  Next ":num -> (num -> ((num -> bool) -> bool))"
  IsTimeOf ":num -> ((num -> bool) -> (num -> bool))"
  TimeOf ":(num -> bool) -> (num -> num)"
  Inf ":(num -> bool) -> bool"
Curried Infixes --
  when ":(num -> *) -> ((num -> bool) -> (num -> *))"
Definitions --
  Next
    |- !t1 t2 f.
        Next t1 t2 f =
        t1 < t2 /\ f t2 /\ (!t. t1 < t /\ t < t2 ==> ~f t)
  IsTimeOf_DEF
    |- IsTimeOf =
       PRIM_REC
       (\f t. f t /\ (!t'. t' < t ==> ~f t'))
       (\g00004 n f t. ?t'. g00004 f t' /\ Next t' t f)
  TimeOf  |- !f n. TimeOf f n = (@t. IsTimeOf n f t)
  when    |- !s p. s when p = (\n. s(TimeOf p n))
  Inf     |- !f. Inf f = (!t. ?t'. t < t' /\ f t')
Theorems --
  IsTimeOf
    |- (!  f t. IsTimeOf 0 f t = f t /\ (!t'. t' < t ==> ~f t')) /\
       (!n f t. IsTimeOf(SUC n)f t = (?t'. IsTimeOf n f t' /\ Next t' t f))
  Inf_EXISTS  |- !f. Inf f ==> (?n. f n)
  Inf_LEAST   |- !f. Inf f ==> LEAST f
  Inf_Next    |- !f. Inf f ==> (!t. f t ==> (?t'. Next t t' f))
  Next_ADD1   |- !f t. f(t + 1) ==> Next t(t + 1)f
  Next_INCREASE
    |- !f t1 t2. ~f(t1 + 1) ==> Next(t1 + 1)t2 f ==> Next t1 t2 f
  Next_IDENTITY
    |- !t1 t2 f. Next t1 t2 f ==> (!t3. Next t1 t3 f ==> (t2 = t3))
  IsTimeOf_TRUE    |- !n f t. IsTimeOf n f t ==> f t
  IsTimeOf_EXISTS  |- !f. Inf f ==> (!n. ?t. IsTimeOf n f t)
```

```
  TimeOf_DEFINED   |- !f. Inf f ==> (!n. IsTimeOf n f(TimeOf f n))
  TimeOf_TRUE      |- !f. Inf f ==> (!n. f(TimeOf f n))
  IsTimeOf_IDENTITY
    |- !n f t1 t2. IsTimeOf n f t1 /\ IsTimeOf n f t2 ==> (t1 = t2)
  TimeOf_INCREASING
    |- !f. Inf f ==> (!n. (TimeOf f n) < (TimeOf f(n + 1)))
  TimeOf_INTERVAL
    |- !f. Inf f ==>
        (!n t. (TimeOf f n) < t /\ t < (TimeOf f(n + 1)) ==> ~f t)
  TimeOf_Next  |- !f. Inf f ==> (!n. Next(TimeOf f n)(TimeOf f(n + 1))f)
************************************----------------------------->%
```

4.4 The Top Level Correctness Goal

The top level correctness goal defines the relation between the two descriptions of SECD, using the abstraction functions. The simplicity of the correctness statement demonstrates the expressive nature of the HOL notation.

4.4.1 attempt1

```
% SECD verification                                                    %
%                                                                     %
% FILE:                attempt1.ml                                    %
%                                                                     %
% DESCRIPTION:  This first effort will try to establish the          %
%               correctness of the cons routine.                      %
%                                                                     %
% USES FILES:   constraints.th, mem_abs.th, when.th                   %
%                                                                     %
% Brian Graham 89.07.21                                              %
%                                                                     %
% Modifications:                                                     %
%                                                                     %
%=====================================================================%
new_theory `attempt1`;;

loadf `load_wordn`;;

map new_parent [`constraints` ;
                `mem_abs`;
                `when`];;

let ftime = ":num"
and mtime = ":num"
and ctime = ":num";;

let fsig = ":^ftime->bool"
and msig = ":^mtime->bool"
and csig = ":^ctime->bool";;

let w9_fvec = ":^ftime->word9"
and w9_mvec = ":^mtime->word9"
and w9_cvec = ":^ctime->word9";;

let w14_fvec = ":^ftime->word14"
and w14_mvec = ":^mtime->word14"
and w14_cvec = ":^ctime->word14";;

let w27_fvec = ":^ftime->word27"
and w27_mvec = ":^mtime->word27"
and w27_cvec = ":^ctime->word27";;

let w32_fvec = ":^ftime->word32"
and w32_mvec = ":^mtime->word32"
and w32_cvec = ":^ctime->word32";;
```

```
let mem14_32 = ":word14->word32";;
let m14_32_mvec = ":^mtime->^mem14_32";;
let M = ":(word14,atom)mfsexp_mem";;
let M_mvec = ":^mtime->^M";;

let state = ":bool # bool";;
let state_msig = ":^mtime->^state"
and state_csig = ":^ctime->^state";;


%================================================================%
% Useful for finding word14 values for memory addresses.         %
%================================================================%
let rem (x,y) = x - y * (x / y);;

letrec mk_bit_list num size =
   (size = 0) => [] |
                 (mk_bit_list (num/2) (size-1))
                  @ [rem (num,2)];;

%================================================================%
% This defines a relation on the inputs and outputs.             %
%================================================================%
let sys_imp =
"(SYS (memory:^m14_32_mvec)
      (SYS_Clocked:^msig)
      (mpc:^w9_mvec)
      (s0:^w9_mvec)          (s1:^w9_mvec)
      (s2:^w9_mvec)          (s3:^w9_mvec)
      (reset_pin:^msig)      (button_pin:^msig)
      (flag0_pin:^msig)      (flag1_pin:^msig)
      (write_bit_pin:^msig) (rmem_pin:^msig)
      (bus_pins:^w32_mvec)  (mar_pins:^w14_mvec)
      (s':^w14_mvec)         (e':^w14_mvec)
      (c':^w14_mvec)         (d':^w14_mvec)
      (free:^w14_mvec)
      (x1:^w14_mvec)         (x2:^w14_mvec)
      (y1:^w14_mvec)         (y2:^w14_mvec)
      (car:^w14_mvec)
      (root:^w14_mvec)       (parent:^w14_mvec)
      (buf1:^w32_mvec)       (buf2:^w32_mvec)
      (arg:^w32_mvec))";;



g "^sys_imp                                    /\
   (reset_constraint reset_pin)                /\
   (clock_constraint SYS_Clocked)              /\
   (free_list_constraint mpc free)             /\
   (valid_program_constraint memory mpc c')
   ==>
    SYS_spec    ((mem_abs o memory) when (is_major_state mpc))
                (s'                 when (is_major_state mpc))
```

```
                (e'                when (is_major_state mpc))
                (c'                when (is_major_state mpc))
                (d'                when (is_major_state mpc))
                (free              when (is_major_state mpc))
                (button_pin        when (is_major_state mpc))
                ((state_abs o mpc) when (is_major_state mpc))
";;
```

Appendix: Abbreviations from hol-init.ml

Several abbreviations are used throughout the source files included in this report. For the ease of the reader, the author's "hol-init.ml" file is included here for reference.

```
let load_all_axioms theory  =
    map (load_axiom theory) (map fst (axioms theory));;

let load_all_definitions theory  =
    map (load_definition theory) (map fst (definitions theory));;

let load_all_theorems theory  =
    map (load_theorem theory) (map fst (theorems theory));;

let load_all theory =
    ( load_all_axioms theory
    ; load_all_definitions theory
    ; load_all_theorems theory
    );;

%----------------------------------------------------------------------------
|                [...] |-? tm[a = b]
|   SYM_TAC   :  ====================
|                [...] |-? tm[b = a]
----------------------------------------------------------------------------%
let SYM_TAC =
   (CONV_TAC (ONCE_DEPTH_CONV SYM_CONV))
   ? failwith 'SYM_TAC';;

%----------------------------------------------------------------------------
|                 |- tm[a = b]
|   SYM_RULE   :  --------------
|                 |- tm[b = a]
----------------------------------------------------------------------------%
let SYM_RULE =
 (CONV_RULE (ONCE_DEPTH_CONV SYM_CONV))
 ? failwith 'SYM_RULE';;


%----------------------------------------------------------------------------
|  Some useful abbreviations of rules, tactics and conversions.
----------------------------------------------------------------------------%
let in_conv       =                  DEPTH_CONV o CHANGED_CONV
and re_conv       =                REDEPTH_CONV o CHANGED_CONV
```

```
and in1_conv      =             ONCE_DEPTH_CONV o CHANGED_CONV
and in_conv_tac   = CONV_TAC  o      DEPTH_CONV o CHANGED_CONV
and re_conv_tac   = CONV_TAC  o    REDEPTH_CONV o CHANGED_CONV
and in1_conv_tac  = CONV_TAC  o ONCE_DEPTH_CONV o CHANGED_CONV
and in_conv_rule  = CONV_RULE o      DEPTH_CONV o CHANGED_CONV
and re_conv_rule  = CONV_RULE o    REDEPTH_CONV o CHANGED_CONV
and in1_conv_rule = CONV_RULE o ONCE_DEPTH_CONV o CHANGED_CONV
;;

let port  = PURE_ONCE_REWRITE_TAC
and prt   = PURE_REWRITE_TAC
and ort   = ONCE_REWRITE_TAC
and rt    = REWRITE_TAC
and poart = PURE_ONCE_ASM_REWRITE_TAC
and part  = PURE_ASM_REWRITE_TAC
and oart  = ONCE_ASM_REWRITE_TAC
and art   = ASM_REWRITE_TAC
;;

let rr    = REWRITE_RULE
and prr   = PURE_REWRITE_RULE
and orr   = ONCE_REWRITE_RULE
and porr  = PURE_ONCE_REWRITE_RULE
;;


%----------------------------------------------------------------------
|  The first assumption which matches the term  tm  is undischarged
|  (c) ISD--Aug.1989.
----------------------------------------------------------------------%
let MATCH_UNDISCH_TAC tm =
  (FIRST_ASSUM \th. UNDISCH_TAC (if  (can (match tm) (concl th))
                                 then (concl th)
                                 else fail
                                ) ? NO_TAC
  )
  ORELSE FAIL_TAC `MATCH_UNDISCH_TAC`;;


%----------------------------------------------------------------------
|  The first assumption which matches the term  tm  is POPped off.
|  (c) ISD--Aug.1989.
----------------------------------------------------------------------%
let MATCH_POP_TAC tm thm_tactic  =
    ( (FIRST_ASSUM \th. UNDISCH_TAC (if   (can (match tm) (concl th))
                                     then (concl th)
                                     else fail
                                    ) ? NO_TAC
      )
      ORELSE FAIL_TAC `MATCH_POP_TAC`
    )
    THEN DISCH_THEN thm_tactic;;
```

```
%----------------------------------------------------------------------------
|  The following lisp calls are to set-up the help facility such that it
|  does NOT use the unix call to "more" for giving help info.
----------------------------------------------------------------------------%
print_string`
.-------------------------------------------------------------.
|   The following four calls to lisp modify the help facility   |
|   such that unix calls to "more" are replaced by "cat".       |
\`-------------------------------------------------------------'
`;;

lisp`
(defun display-file (fname)
   (system (uconcat "cat " fname))
)`;;

lisp`
(defun ml-help (tok)
   (if (fileexists 'help tok)
       (display-file (find-file(fileof 'help tok)))
       (msg-failwith "help" "No information available on " tok)
   )
)`;;

lisp`
(defun keyword-search (key fname)
   (system (uconcat "fgrep '" key "' " fname))
)`;;

lisp`
(defun ml-keyword (tok)
   (keyword-search tok (find-file(fileof 'kwic 'hol)))
)`;;
```

Bibliography

[Gra89] Brian Graham. SECD: Design Issues. to be published as a Research Report,Computer Science Department, University of Calgary, 1989.

[Mas88] Ian A. Mason. Verification of Programs that Destructively Manipulate Data. *Science of Computer Programming*, 10:177–210, 1988.