MIPS Open Developer Day

Saraj Mudigonda
Yuri Panchul
Daniel Bowman
Siobhan Lyons

Agenda

- 12:30 - 1:15pm Welcome & Introduction
- 1:15 - 1:45pm Demo: MIPS Components in Action
- 1:45 - 2:00pm Break
- 2:00 - 3:30pm Hands-on Labs & Exercises
- 3:30 - 3:45pm Break
- 3:45 - 4:45pm Build Your Own MIPS-based SoC
- 4:45 - 5:15pm Present Your innovation
- 5:15 - 5:30pm Wrap-up

Housekeeping

- Workshop Prerequisites
    - Sign up and activate a MIPS Open account - CLICK HERE
    - Accept the License Agreement and request the MIPS Open FPGA package in the downloads section - CLICK HERE
    - A Windows or Linux notebook
- Loaner Hardware
    - Altera/Xilinx FPGA Boards
    - USB Hub
    - Cables
    - SSD Drive
- What is included in SSD Drive?
    - MIPS Open FPGA – Developer Day package
    - Altera Quartus and Xilinx Vivado tools
- Housekeeping
    - Please delete the Altera Quartus and Xilinx Vivado tools on the SSD!!!
    - When you leave the class, please leave the FPGA boards, cables and USB hub.
    - The SSD is yours to take home

WAVE®
COMPUTING

Welcome & Introduction

MIPS Open™

MIPS Open Milestones

Dec 17th, 2018

Wave Computing Launches MIPS Open Initiative

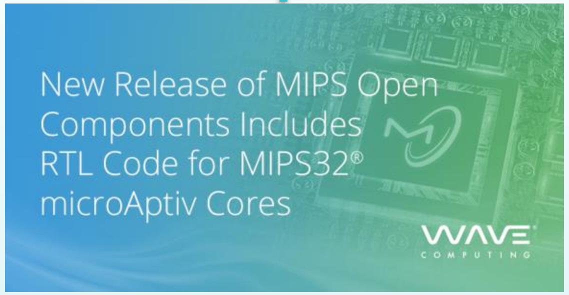
Feb 19th, 2019

Wave Computing + prpl Foundation partner on MIPS Open Advisory Board

March 28th, 2019

Wave's First MIPS Open Program Components Now Available

May 13th, 2019



New Release of MIPS Open Components Includes RTL Code for MIPS32® microAptiv Cores

May 30th, 2019

Wave Computing and Imperas Introduce New MIPS Open Simulator—MIPSOpenOVPsim™

MIPS Open Components

MIPS Open Tools

- Integrated Development Environment
- Includes both Embedded RTOS & Linux editions
- Extensible framework for 3rd party tools

MIPS Open FPGA

- Non-production implementation for teaching, training, evaluation
- 25 hands-on labs
- Preconfigured basic microAptiv core

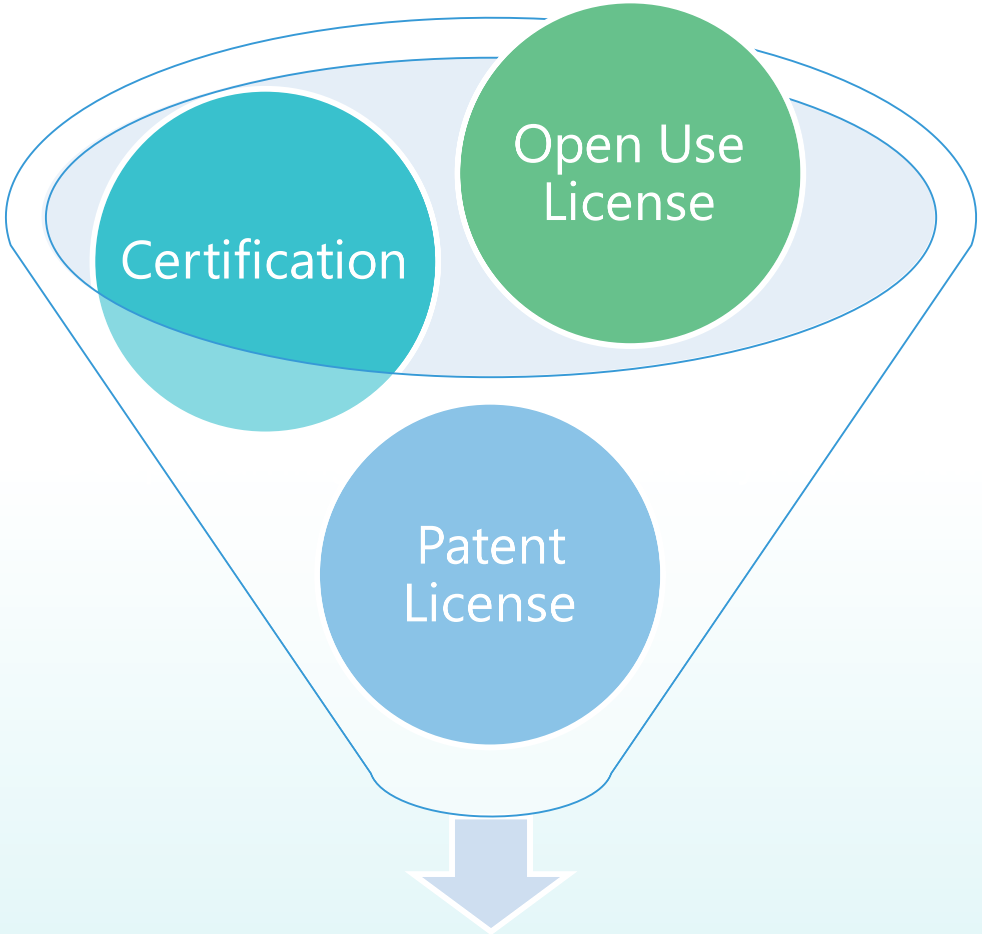
MIPS Open Cores

- Verilog Source RTL Package
- Includes microAptiv Microprocessor & Microcontroller cores
- Ideal for battery-powered applications

MIPS Open Architecture

- Latest Release 6 Architecture that includes both 32-/64-bit ISA and extensions – DSP, SIMD, VZ, MT for multimedia, automotive, IoT, modem applications

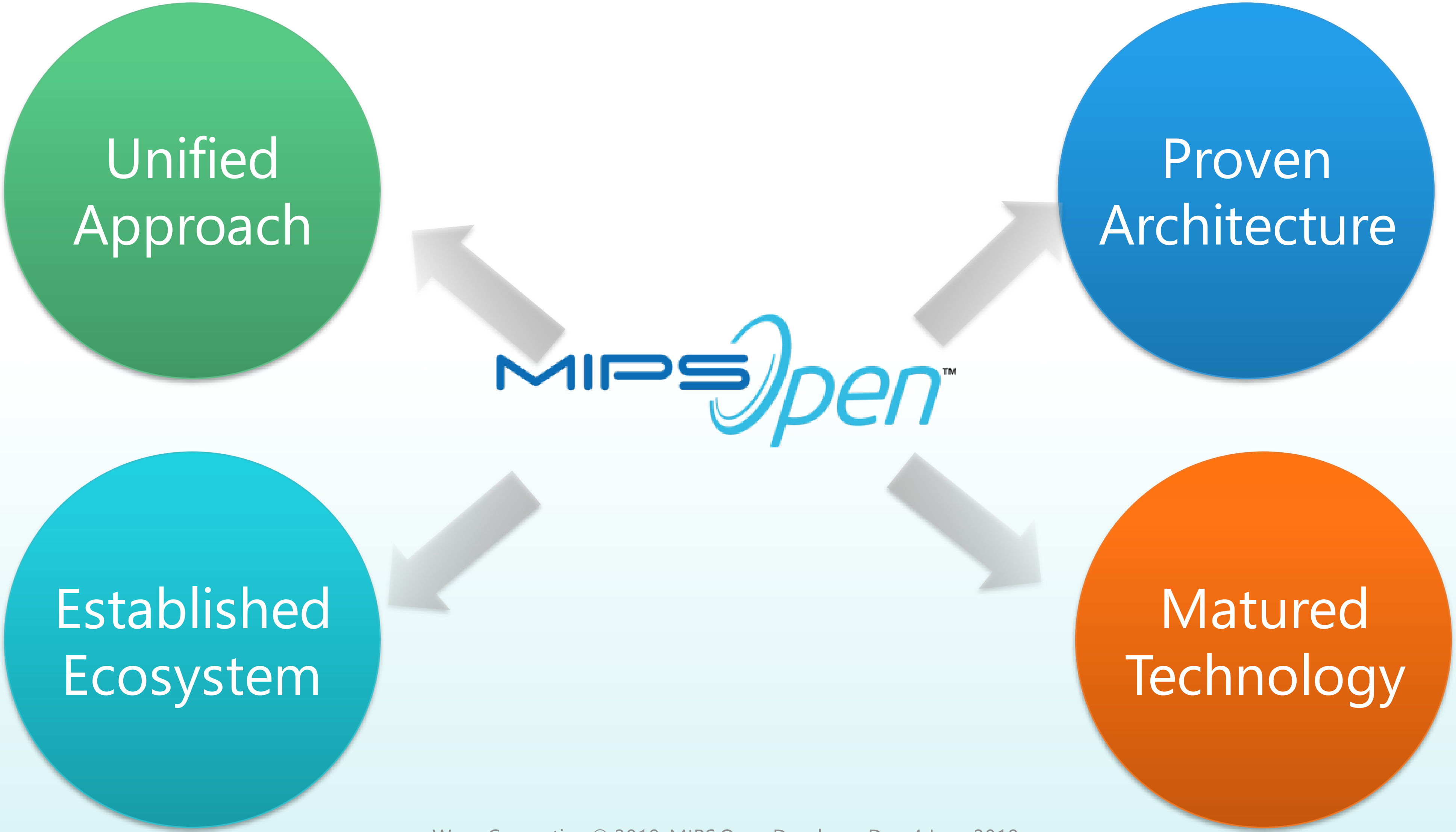
MIPS Open Features



Comprehensive Package

- No license fees, no royalties, non-exclusive, worldwide license
- Latest MIPS R6 architecture, microAptiv cores, Tools
- Right and license under R6 architecture patents to design, build and sell cores
- Use of the "MIPS Certified" trademark logo for certified cores

MIPS Open Benefits



MIPS Open Advisory Committee Membership Levels

Individual Membership

Free per year

- Allows participation in all working groups
- Represent individual academic and non-profits

Silver Membership

$10,000 per year

- Can be appointed to lead a working group
- Vote as a Silver class representation on the board of representations

Gold Membership

$50,000 per year

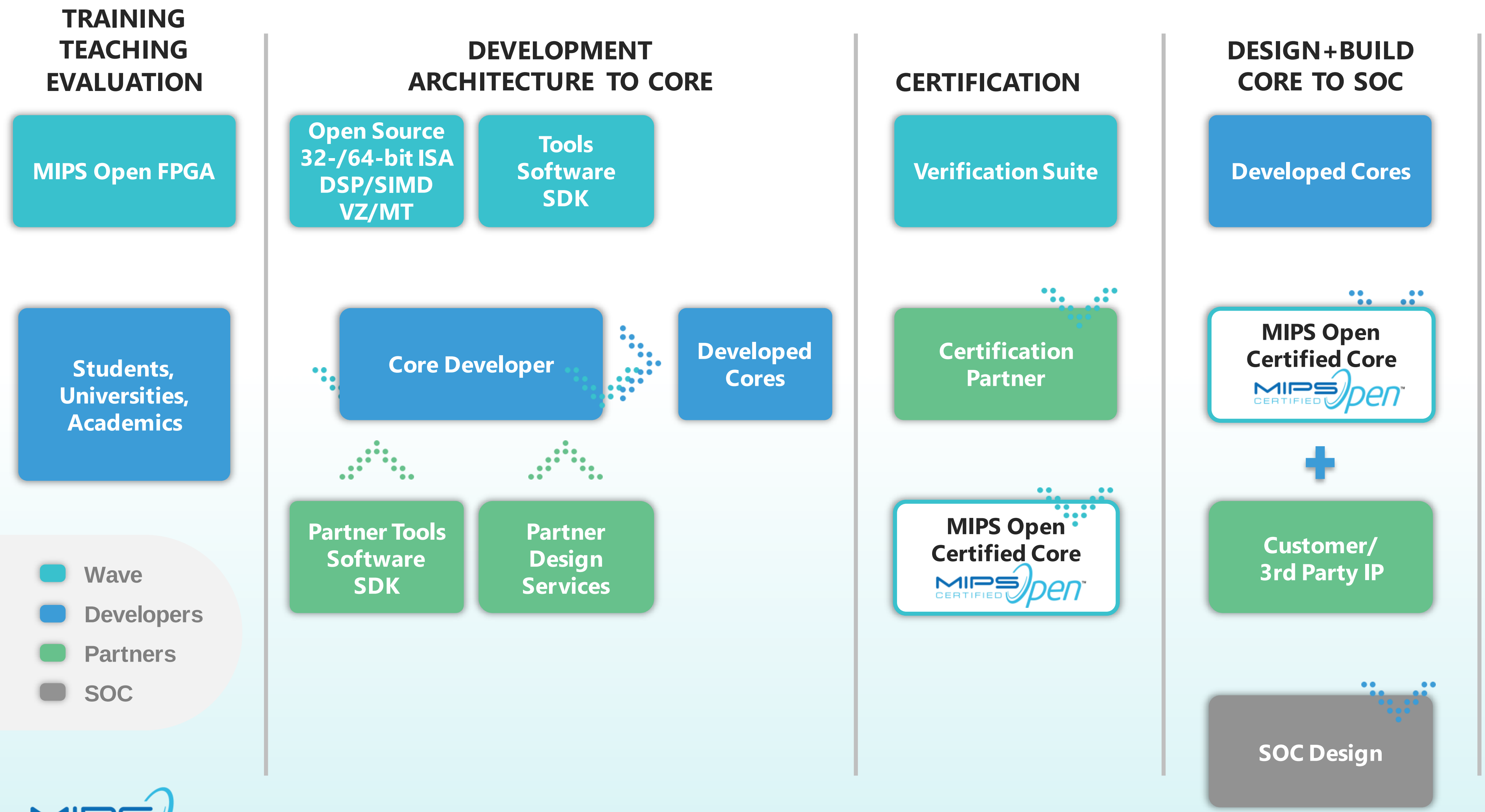
- Can be appointed to lead a working group
- Vote as a Gold class representation on the board of representations

Platinum Membership

$100,000 per year

- Leadership decision-making level of membership
- Automatically appointed to the board of directors
- Set direction, approve budgets, and projects

MIPS Open™ Development



TRAINING TEACHING EVALUATION

MIPS Open FPGA

Students, Universities, Academics

DEVELOPMENT ARCHITECTURE TO CORE

Open Source 32-/64-bit ISA DSP/SIMD VZ/MT

Tools Software SDK

Core Developer

Developed Cores

Partner Tools Software SDK

Partner Design Services

CERTIFICATION

Verification Suite

Certification Partner

MIPS Open Certified Core

DESIGN+BUILD CORE TO SOC

Developed Cores

MIPS Open Certified Core

+

Customer/ 3rd Party IP

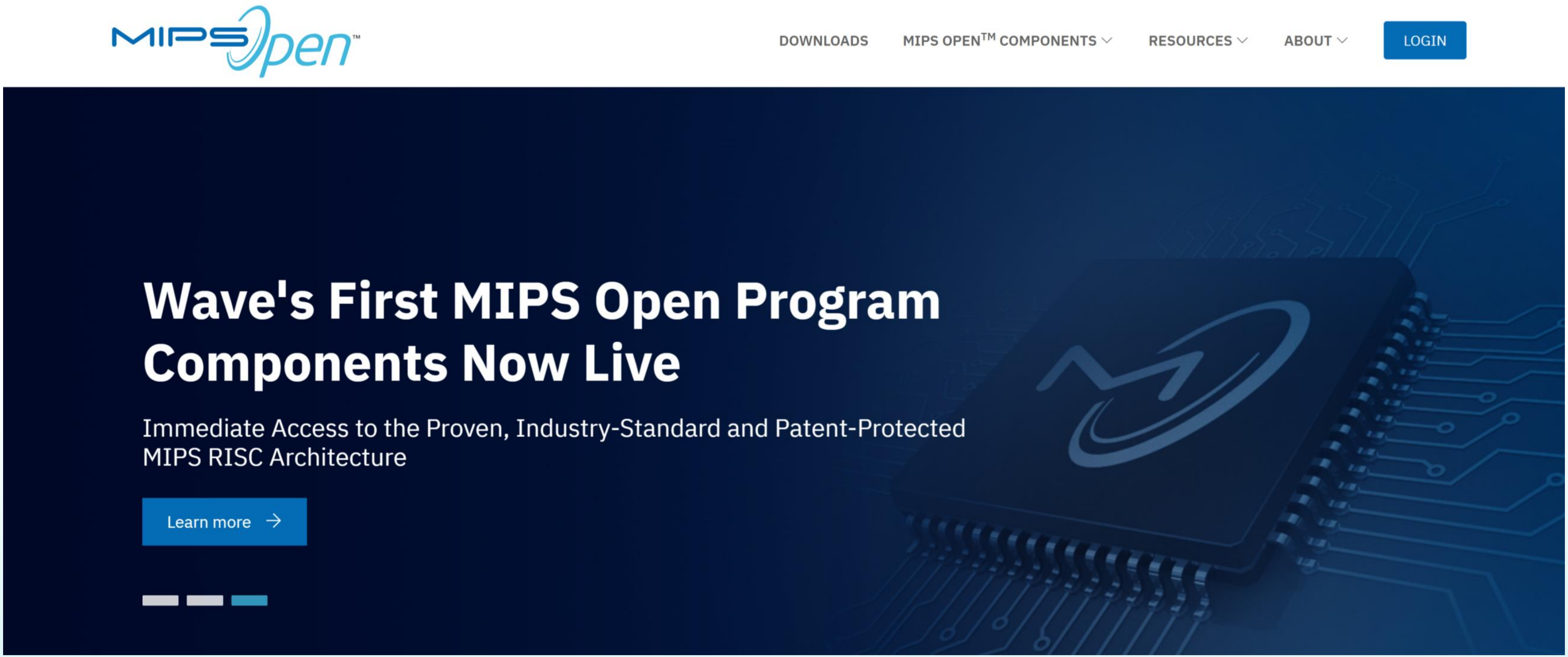
SOC Design

SELL

SOC Design

- ■ Wave
- ■ Developers
- ■ Partners
- ■ SOC

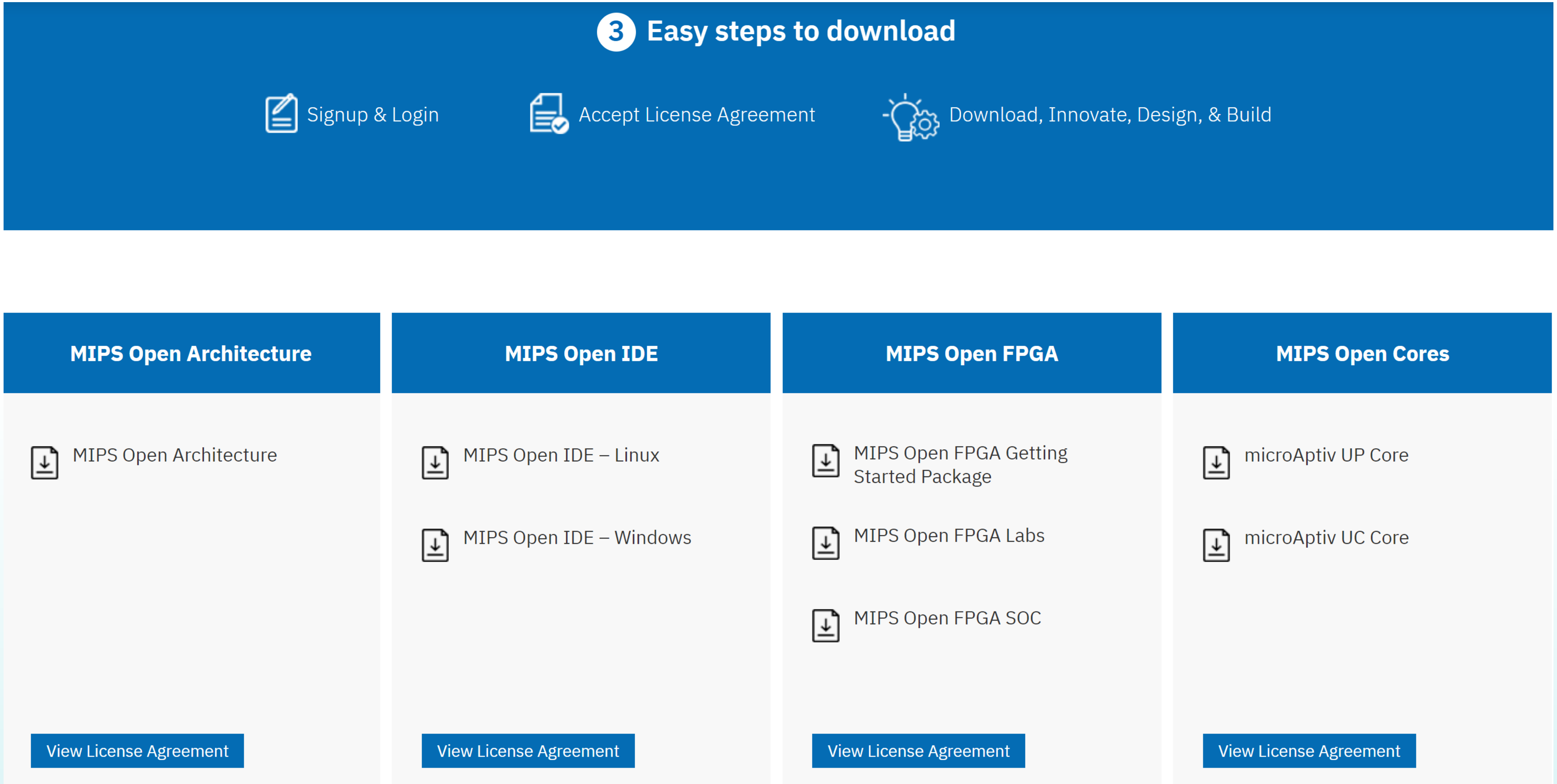
MIPSOpen.com Now Live



Wave's First MIPS Open Program Components Now Live

Immediate Access to the Proven, Industry-Standard and Patent-Protected MIPS RISC Architecture

3 Easy Steps to Download, Innovate, Design & Build



3 Easy steps to download

Signup & Login

Accept License Agreement

Download, Innovate, Design, & Build

MIPS Open Architecture

- MIPS Open Architecture

View License Agreement

MIPS Open IDE

- MIPS Open IDE – Linux
- MIPS Open IDE – Windows

View License Agreement

MIPS Open FPGA

- MIPS Open FPGA Getting Started Package
- MIPS Open FPGA Labs
- MIPS Open FPGA SOC

View License Agreement

MIPS Open Cores

- microAptiv UP Core
- microAptiv UC Core

View License Agreement

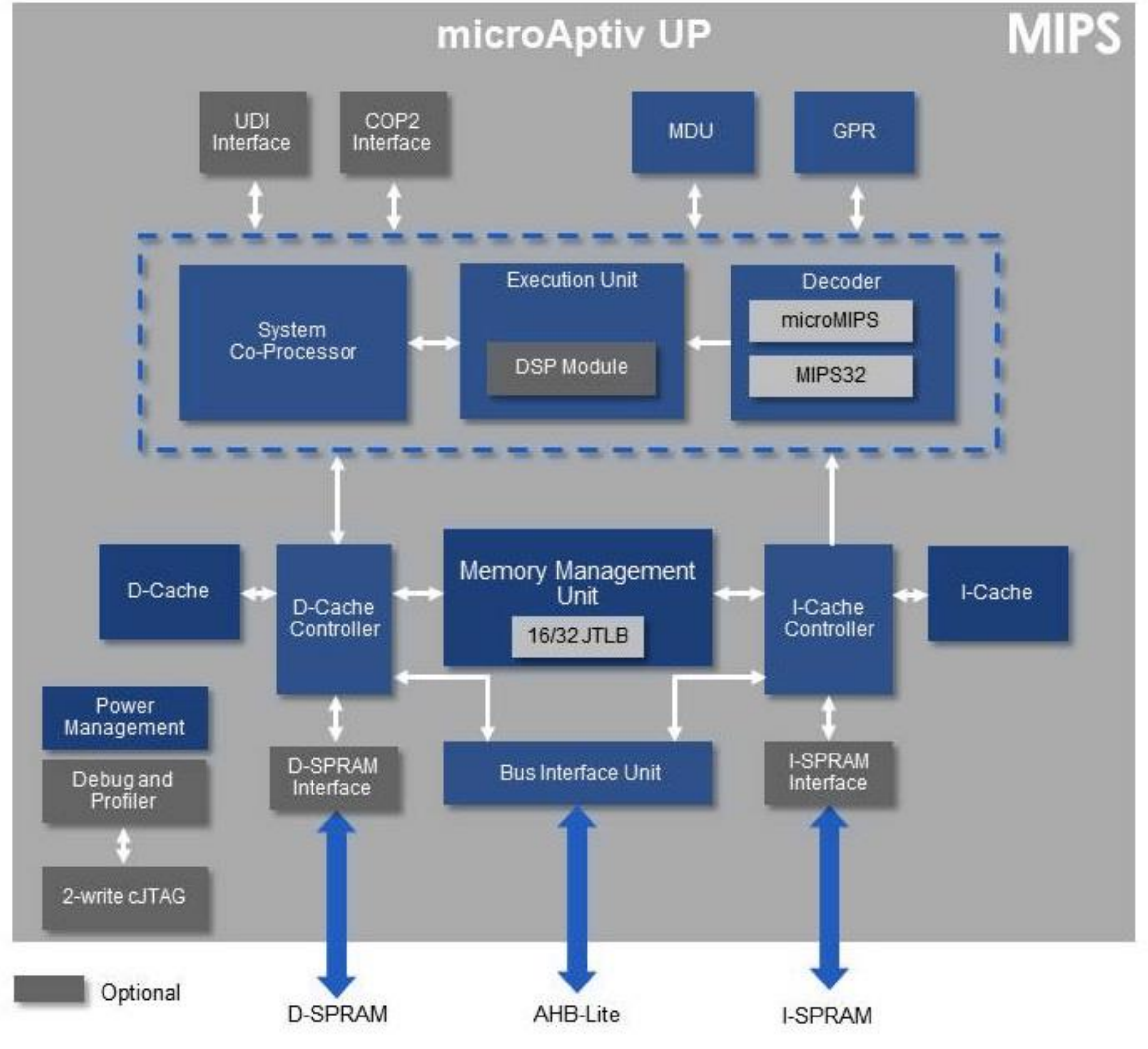
MIPSOpen™ Open Use cores

MIPS Leading in its class Performance efficient microAptiv Cores

- Improved 5 stage pipeline architecture
- 32 GPRs, with up to 16 shadow register sets
- Minimal interrupt latency
- Integrated DSP ASE outperfroms Cortex-M4
- 3.5 CoreMarks/MHz, 1.7DMIPs/MHz

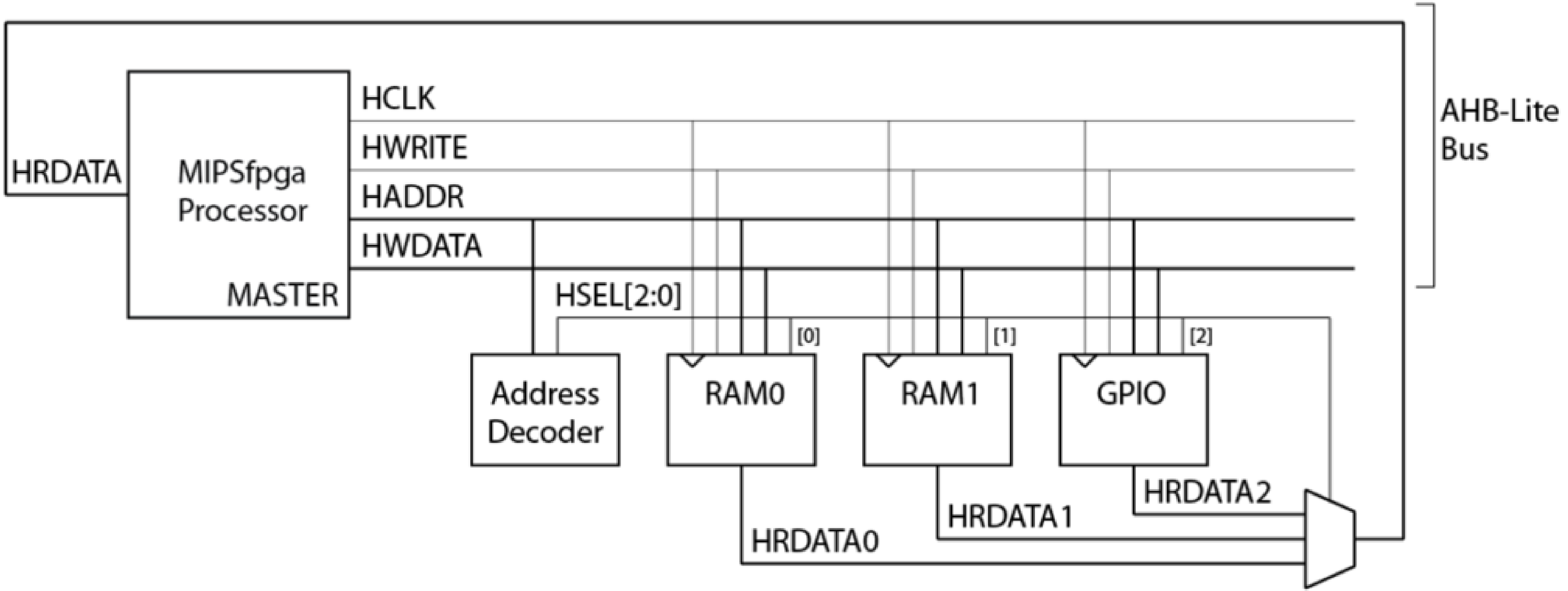
Higher performance & scalability

- Shadow Registers for faster context switching
- Mostly single operation instructions
- Simpler memory addressing modes
- User Defined Instructions (UDI) for custom ISA



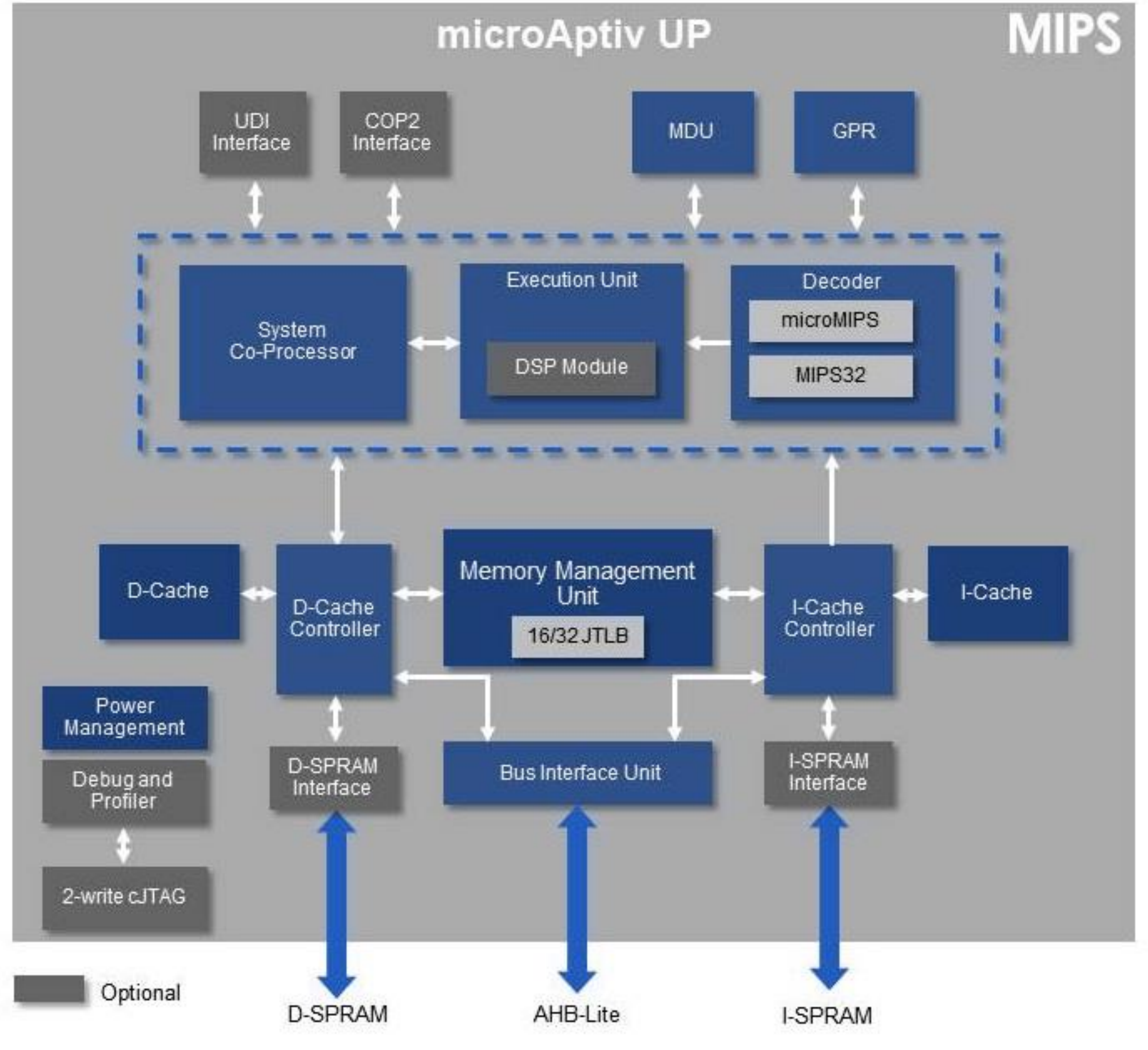
Demo: MIPS Components in Action

High-level view of MIPS Open FPGA system

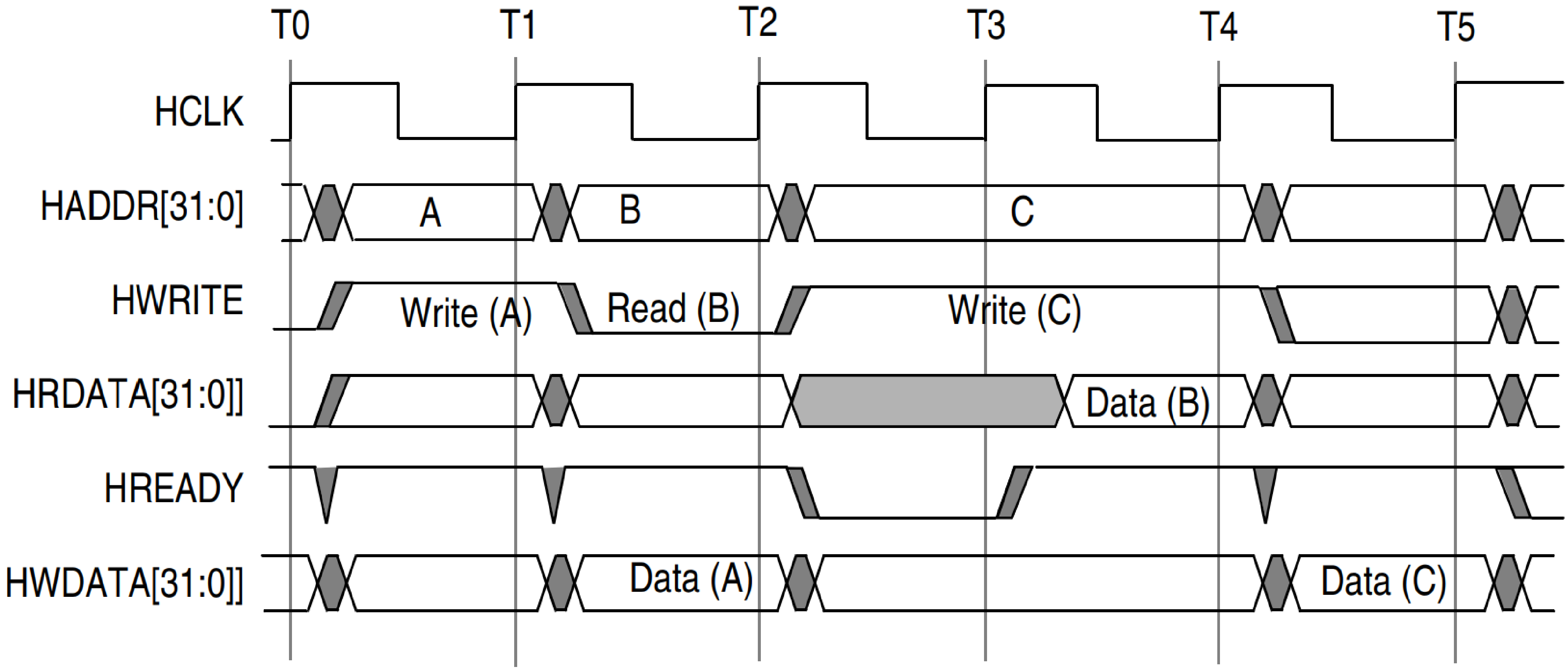


MIPS microAptiv UP and its interface options

- System bus, AHB-Lite, with optional bridge to AXI
- CorExtend / UDI - User-Defined Instructions
- Cop2 - an older, more flexible and complicated coprocessor interface
- Data ScratchPad RAM, DSPRAM
  - Custom block, can be used as fixed-latency memory or high-speed I/O
- Instruction ScratchPad RAM, ISPRAM



MIPS Open FPGA system bus uses AHB-Lite



Other components of MIPS Open FPGA system

- Serial loader
  - A hardware block that receives a file in Motorola S-Record format, parses it using state machine and writes into a system memory
- Slow clock for run-time debug
  - A clock divider that allows running the processors with few Hertz frequency
    - Useful to observe cycle behavior of the processor in real time
- External SDRAM memory controller
- External interrupt controller
- Extra wiring to support labs to observe cache and CPU pipeline bypasses

Take a Break

Hands-on Labs & Exercises
The workflow

Demo & Hands-on Labs

- programs/01_light_sensor
- programs/02_interrupts
- programs/03_cache_misses
- programs/04_pipeline_bypasses

Housekeeping

- For this workshop you will need to connect 3 USB devices (SSD drive, FPGA download cable, USB-to-UART serial cable). We have a 4 port USB hub if you need one.
- Connect the drive to USB port
- Reboot or start your laptop
- Hit key F12 prior to booting your normal OS
- Select external USB drive as the new source
- You now have Lubuntu loaded with Intel FPGA Quartus and Xilinx Vivado tools

Basic usage using command line

- cd ~/mipsopen/boards/de10_lite (or another board)
- make all load
- Press reset (or KEY 0 on some boards) to reset the processor
- The default hardcoded program should start to work
- cd ~/mipsopen/programs/00_counter (or other program)
- make program srecord uart
- If computer uses serial connection other than ttyUSB0 (the default), then:
  - make program srecord uart UART=1 (or 2, 3, etc)
- The program uploaded via USB-to-UART is now running

Basic usage using synthesis GUI

- cd ~/mipsopen /boards/de10_lite (or another board)
- For Intel FPGA boards, run ./make_project.sh to create a scratch directory *project*
- Run synthesis and FPGA configuration in scratch directory
- Press reset (or KEY 0 on some boards) to reset the processor.
- The default hardcoded program should start to work.
- cd ~/mipsopen /programs/00_counter (or other program)
- make program srecord uart [UART=0,1,2…]
- The program uploaded via USB-to-UART is now running.

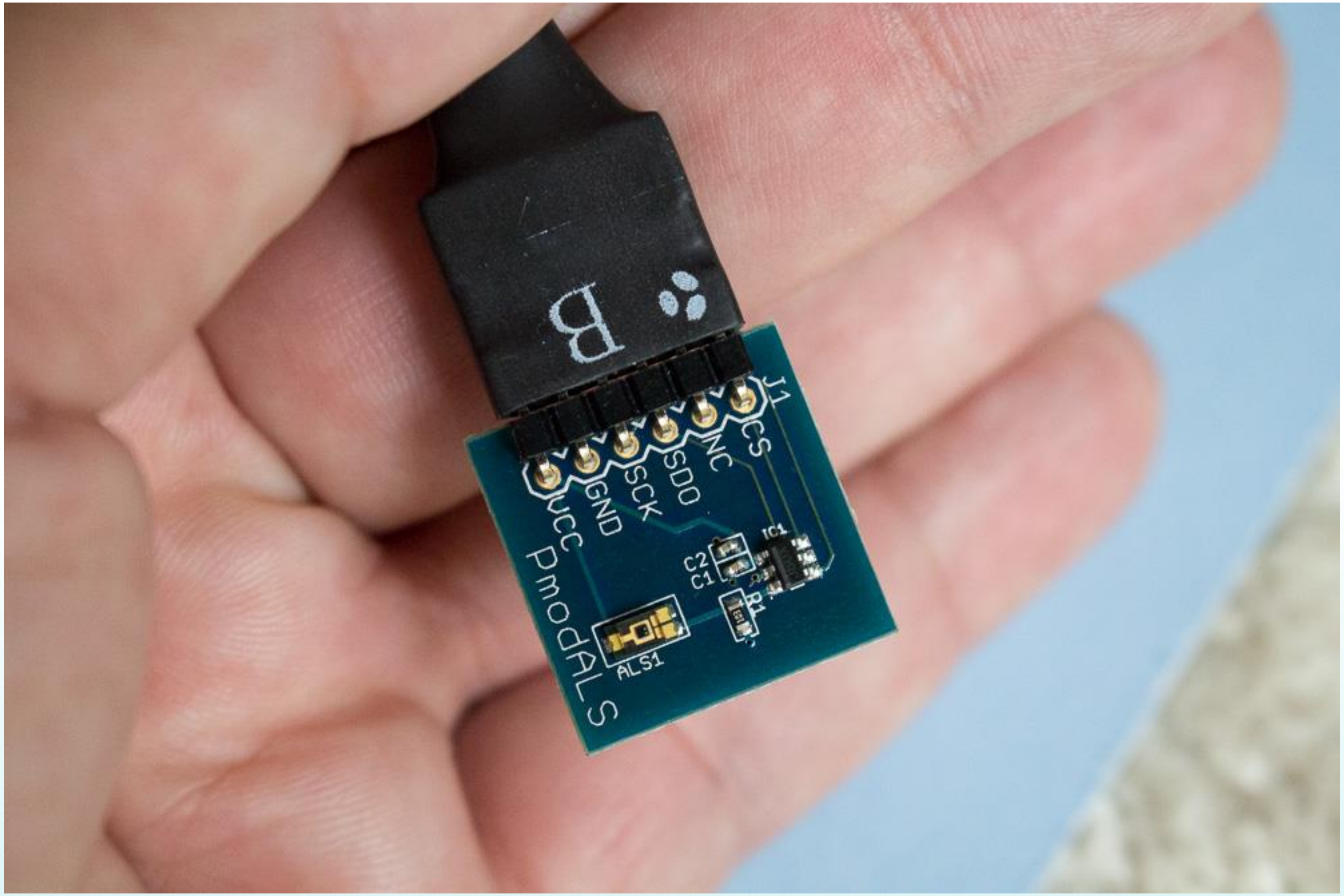
WAVE®
COMPUTING

Integration with Light Sensor

programs/01_light_sensor

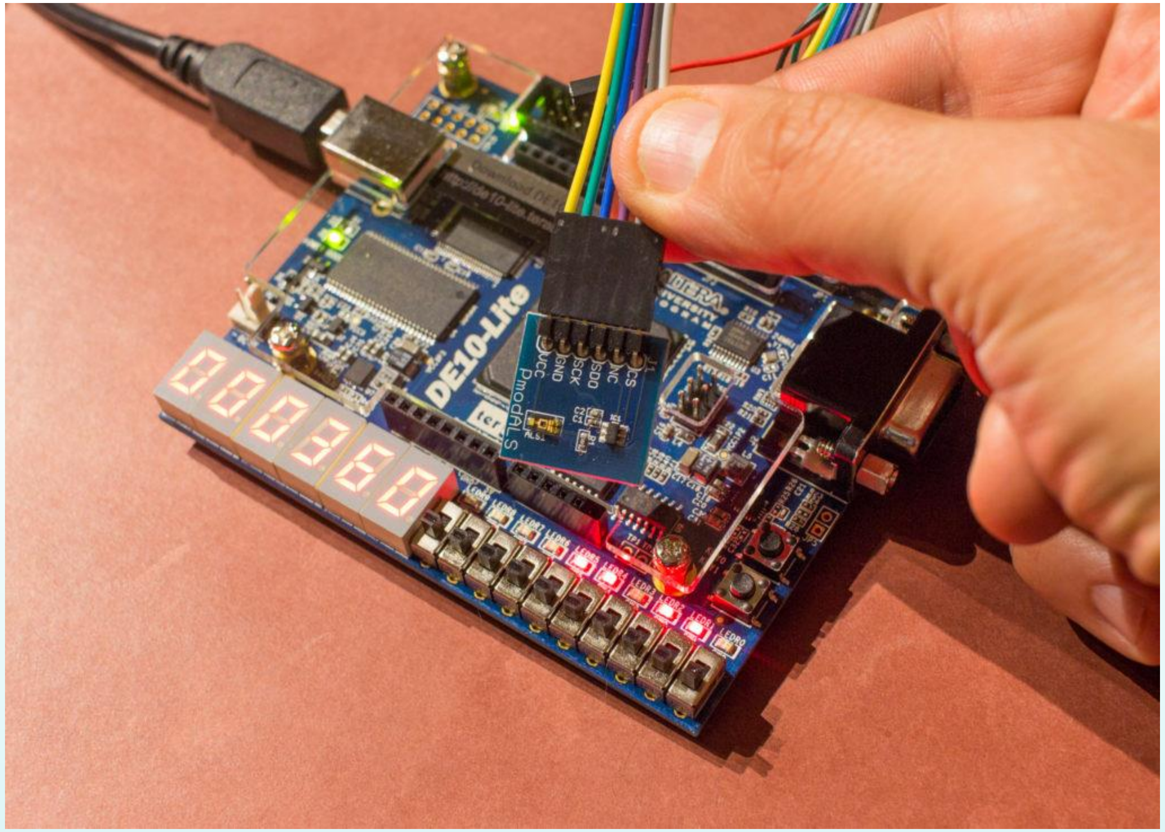
MIPS Open™

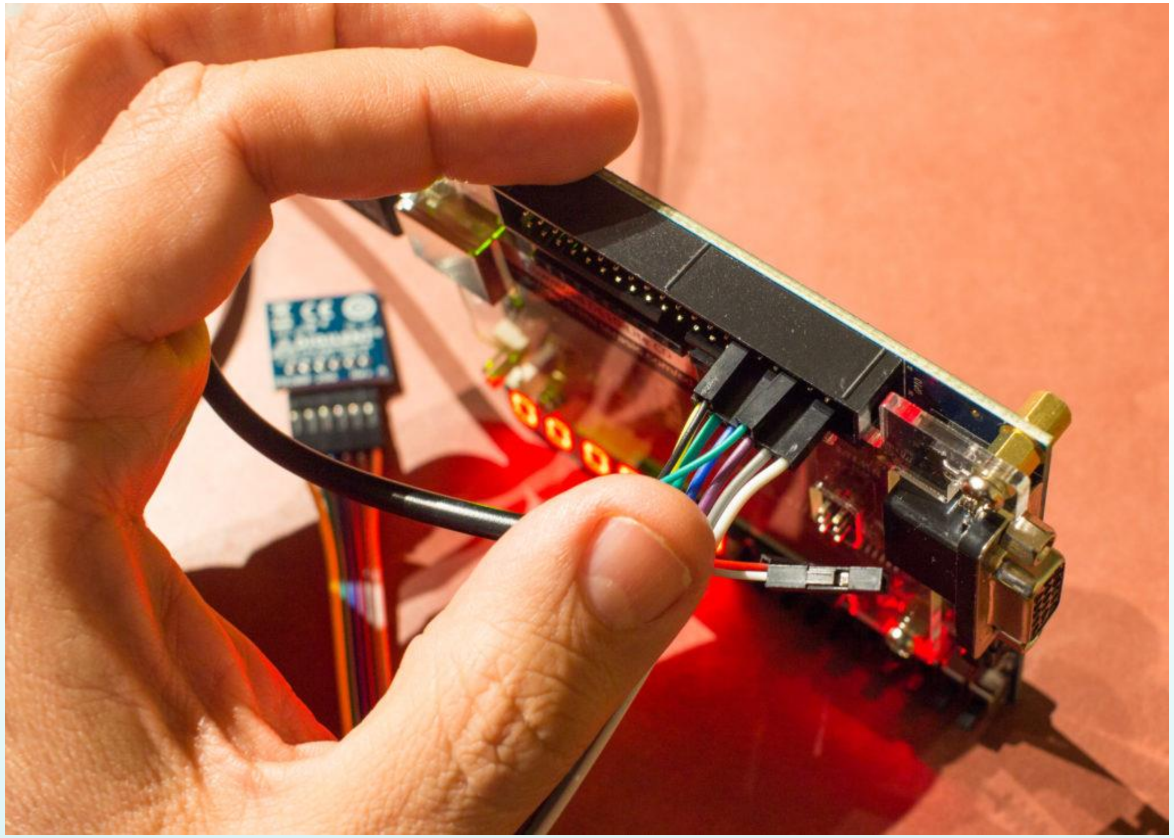
Digilent PmodALS - Ambient Light Sensor



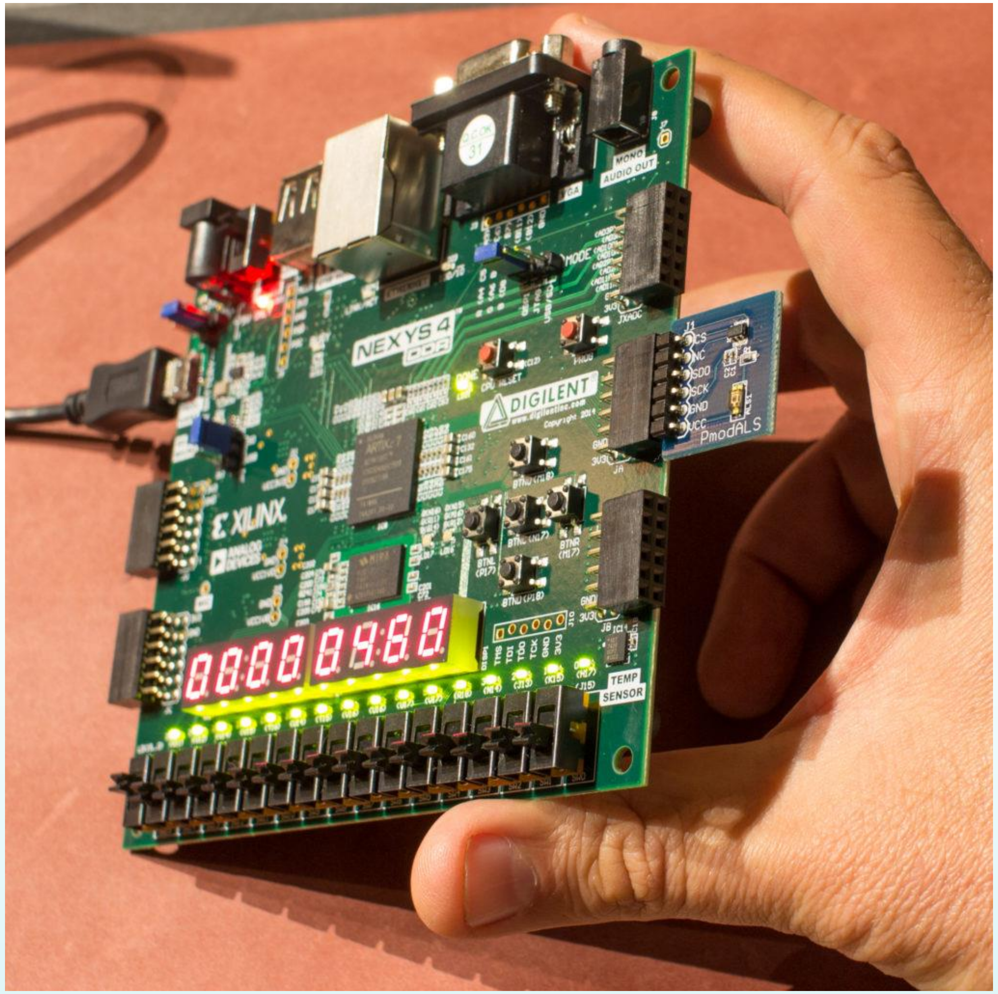
Connecting Light Sensor to Terasic DE10-Lite

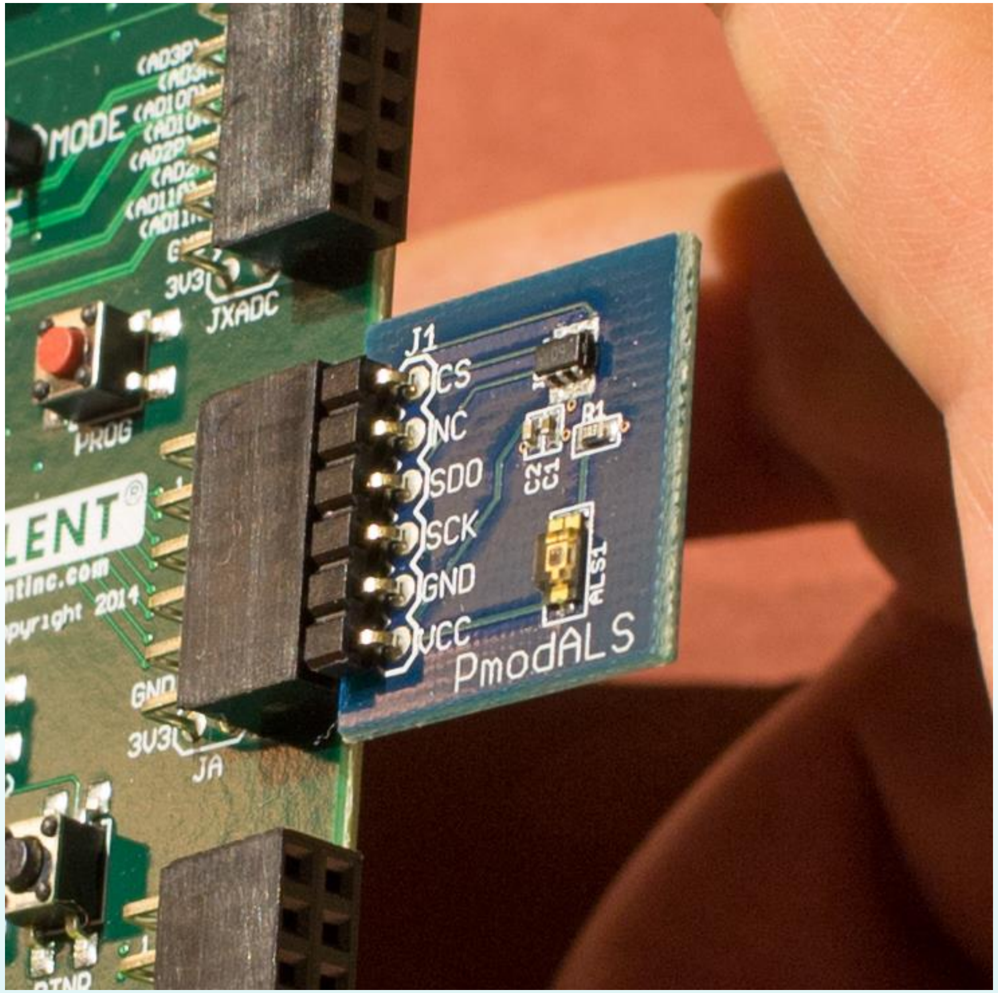
USB-to-UART (needed to upload the program into MIPSfpga SoC): green TX jumper goes into 3rd pin from upper right corner, black GND jumper goes into 6th pin from upper right corner. Light Sensor is connected to the second row of pins as shown with color-coded jumpers.





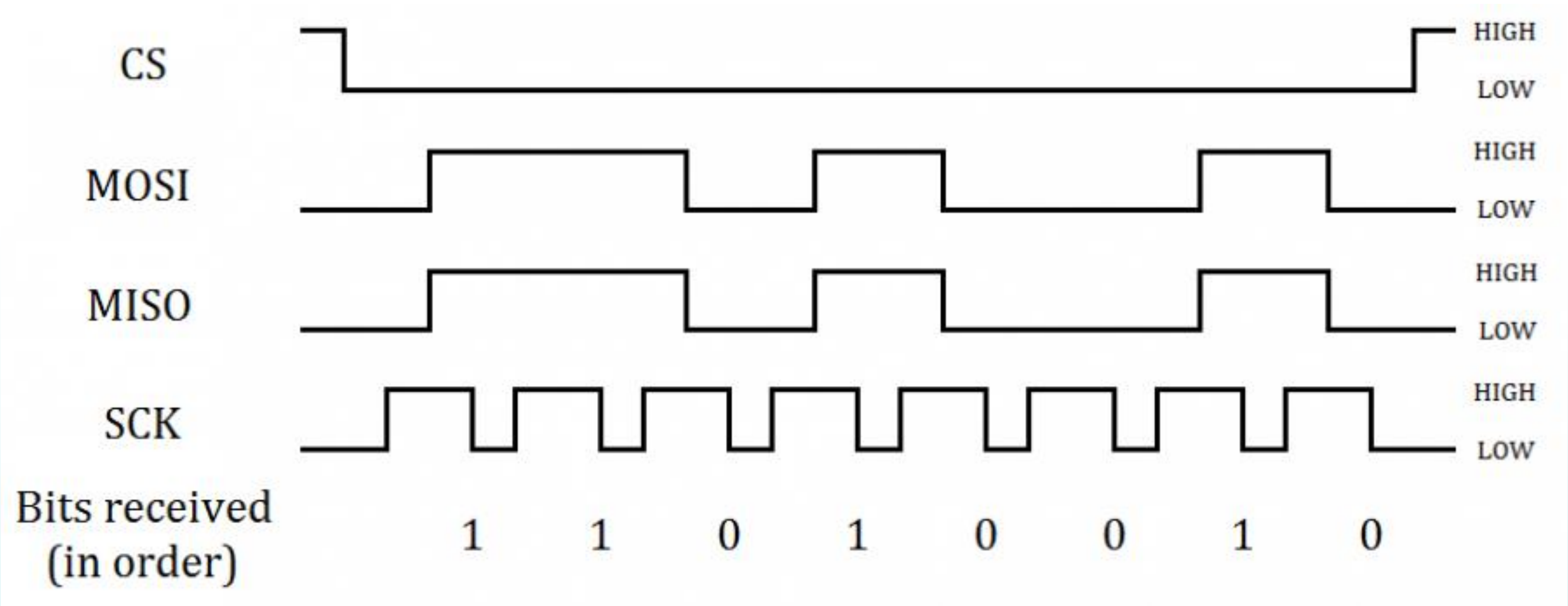
Connecting Light Sensor to Digilent Nexys A7





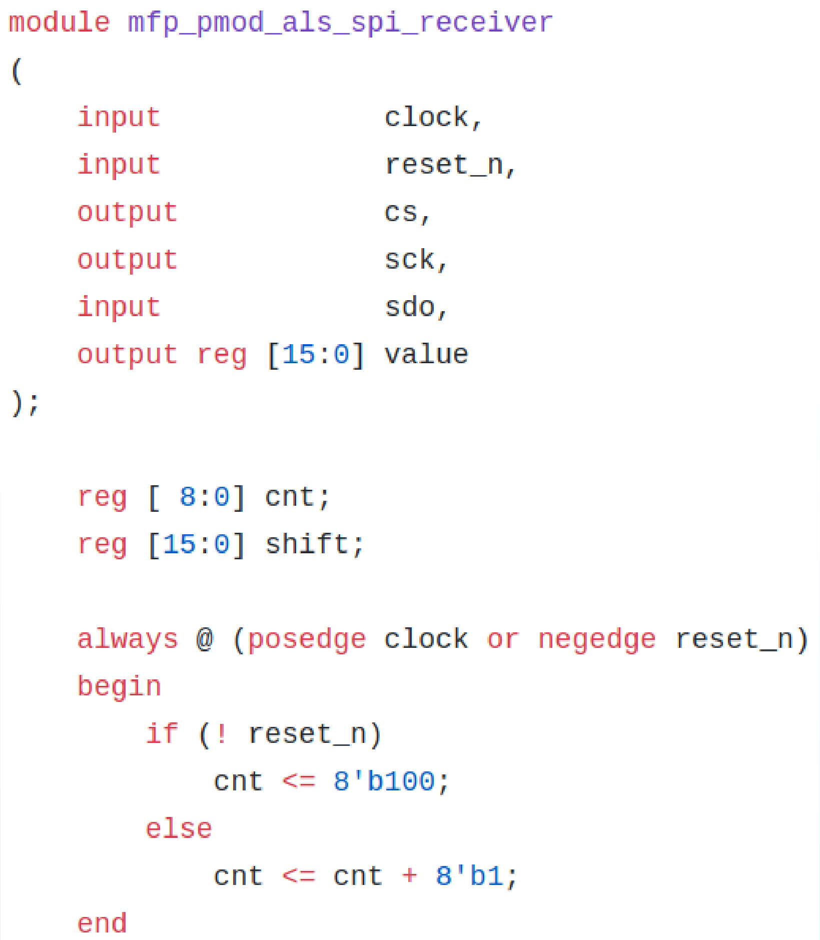
SPI Protocol

https://reference.digilentinc.com/pmod:communication_protocols:spi

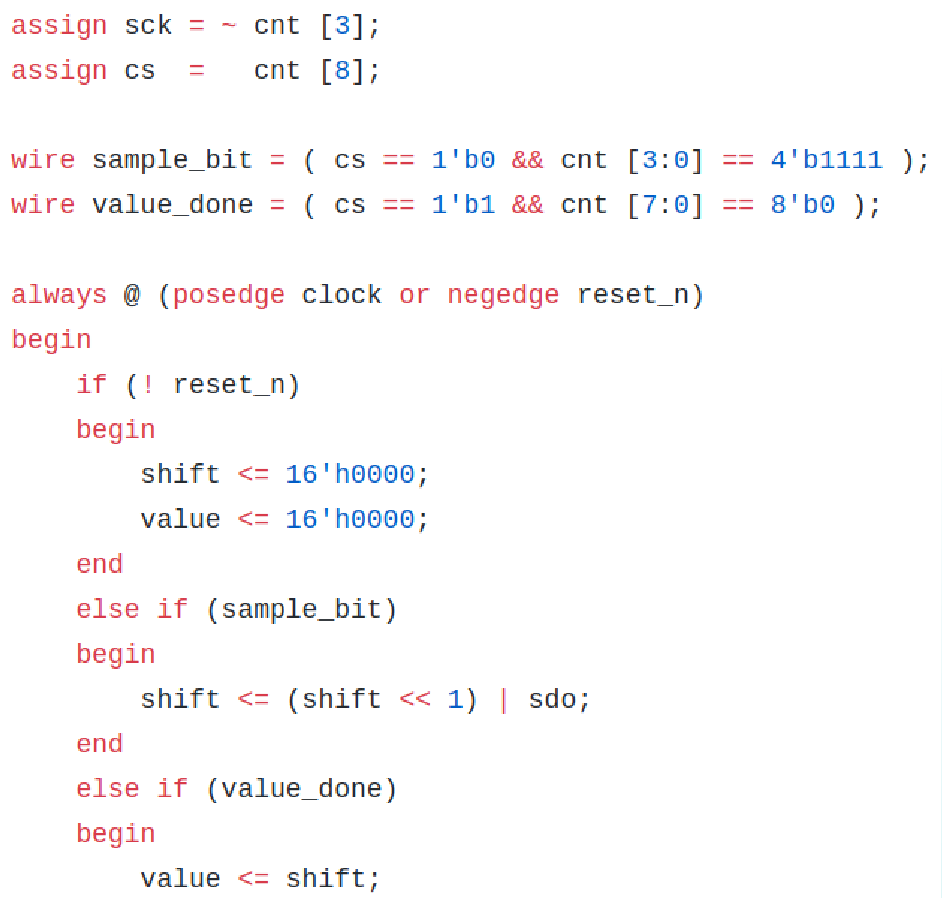


Light Sensor SPI interface module

system_rtl/mfp_pmod_als_spi_receiver.v



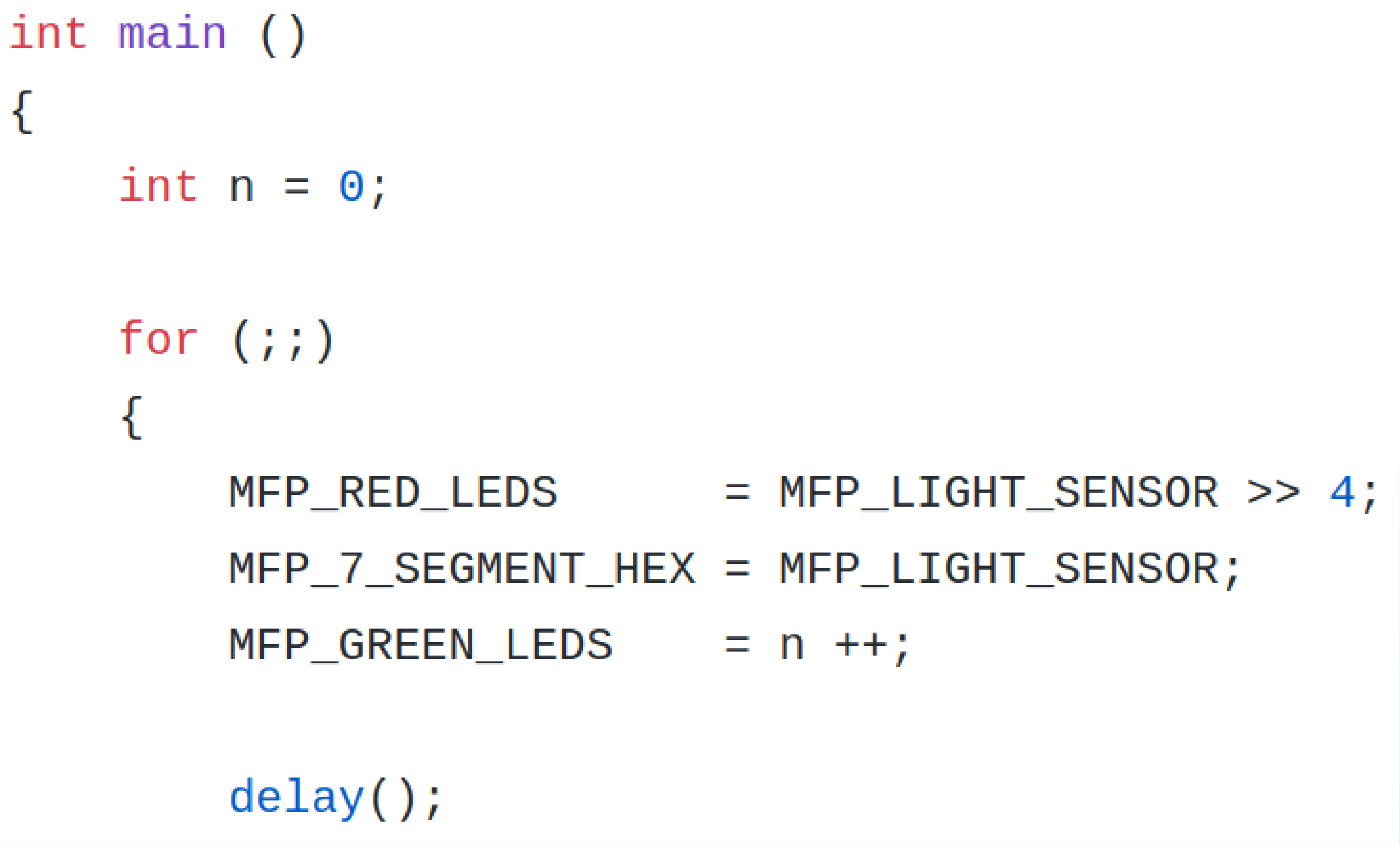
```
module mfp_pmod_als_spi_receiver
(
    input                 clock,
    input                 reset_n,
    output                cs,
    output                sck,
    input                 sdo,
    output reg [15:0] value
);

    reg [ 8:0] cnt;
    reg [15:0] shift;

    always @ (posedge clock or negedge reset_n)
    begin
        if (! reset_n)
            cnt <= 8'b100;
        else
            cnt <= cnt + 8'b1;
    end
```



```
assign sck = ~ cnt [3];
assign cs  =   cnt [8];

wire sample_bit = ( cs == 1'b0 && cnt [3:0] == 4'b1111 );
wire value_done = ( cs == 1'b1 && cnt [7:0] == 8'b0 );

always @ (posedge clock or negedge reset_n)
begin
    if (! reset_n)
    begin
        shift <= 16'h0000;
        value <= 16'h0000;
    end
    else if (sample_bit)
    begin
        shift <= (shift << 1) | sdo;
    end
    else if (value_done)
    begin
        value <= shift;
```

Program that does memory-mapped I/O

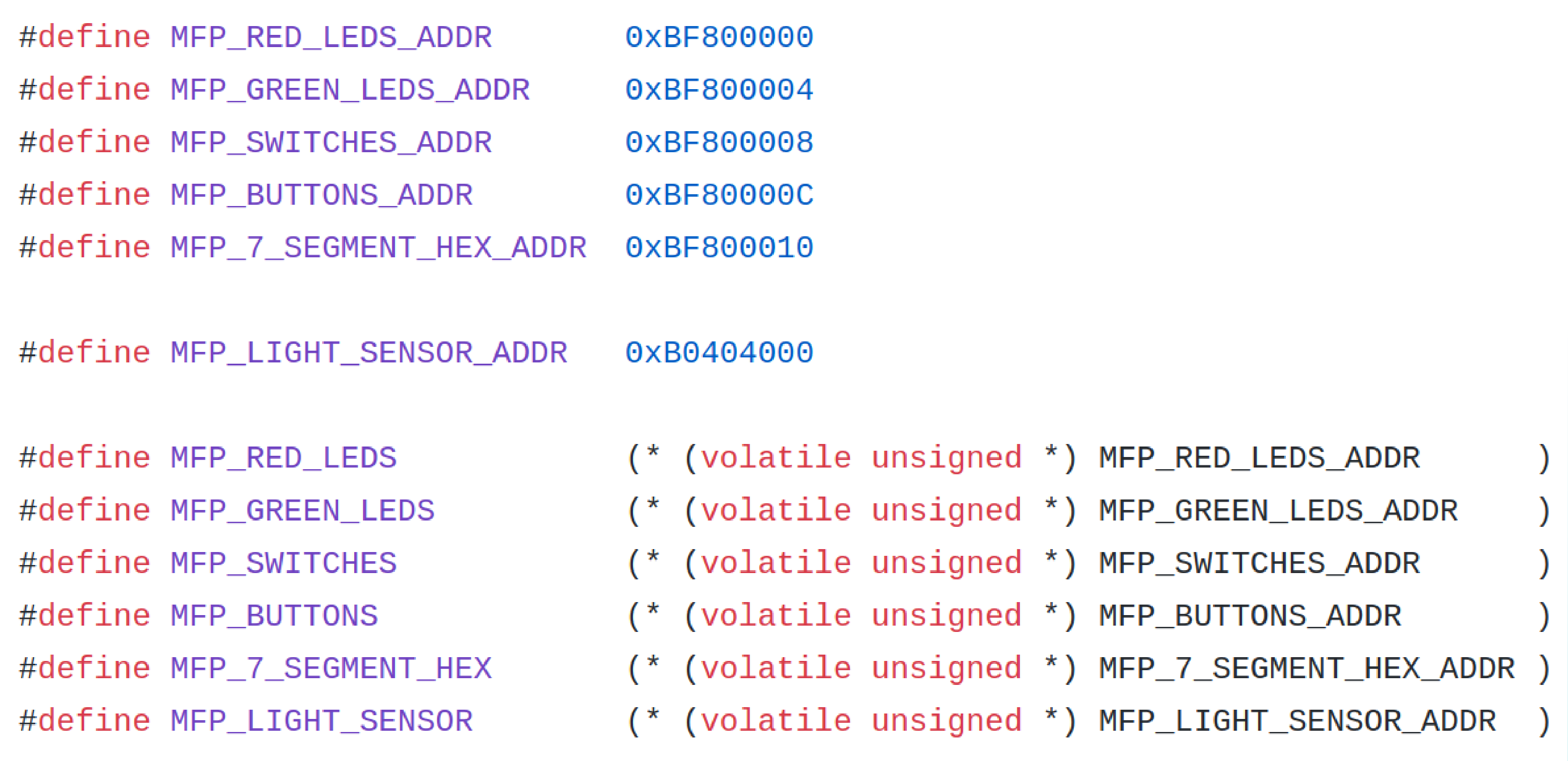
programs/01_light_sensor/main.c



```
int main ()
{
    int n = 0;

    for (;;)
    {
        MFP_RED_LEDS       = MFP_LIGHT_SENSOR >> 4;
        MFP_7_SEGMENT_HEX = MFP_LIGHT_SENSOR;
        MFP_GREEN_LEDS     = n ++;

        delay();
```

Header that defines uncached I/O addresses

programs/01_light_sensor/mfp_memory_mapped_registers.h



```c
#define MFP_RED_LEDS_ADDR       0xBF800000
#define MFP_GREEN_LEDS_ADDR     0xBF800004
#define MFP_SWITCHES_ADDR       0xBF800008
#define MFP_BUTTONS_ADDR        0xBF80000C
#define MFP_7_SEGMENT_HEX_ADDR  0xBF800010

#define MFP_LIGHT_SENSOR_ADDR   0xB0404000

#define MFP_RED_LEDS            (* (volatile unsigned *) MFP_RED_LEDS_ADDR      )
#define MFP_GREEN_LEDS          (* (volatile unsigned *) MFP_GREEN_LEDS_ADDR    )
#define MFP_SWITCHES            (* (volatile unsigned *) MFP_SWITCHES_ADDR      )
#define MFP_BUTTONS             (* (volatile unsigned *) MFP_BUTTONS_ADDR       )
#define MFP_7_SEGMENT_HEX       (* (volatile unsigned *) MFP_7_SEGMENT_HEX_ADDR )
#define MFP_LIGHT_SENSOR        (* (volatile unsigned *) MFP_LIGHT_SENSOR_ADDR  )
```

To integrate software and hardware together

- programs/01_light_sensor/main.c
- programs/01_light_sensor/mfp_memory_mapped_registers.h
- system_rtl/mfp_pmod_als_spi_receiver.v
- system_rtl/mfp_ahb_lite_pmod_als.v
- system_rtl/mfp_ahb_lite_matrix_config.vh
- system_rtl/mfp_ahb_lite_matrix.v
- system_rtl/mfp_ahb_lite_matrix_with_loader.v
- system_rtl/mfp_system.v
- boards/de10_lite/de10_lite.v
- boards/nexys4_ddr/nexys4_ddr.v

Other sensors from Digilent

Keyboard

Rotary encoder

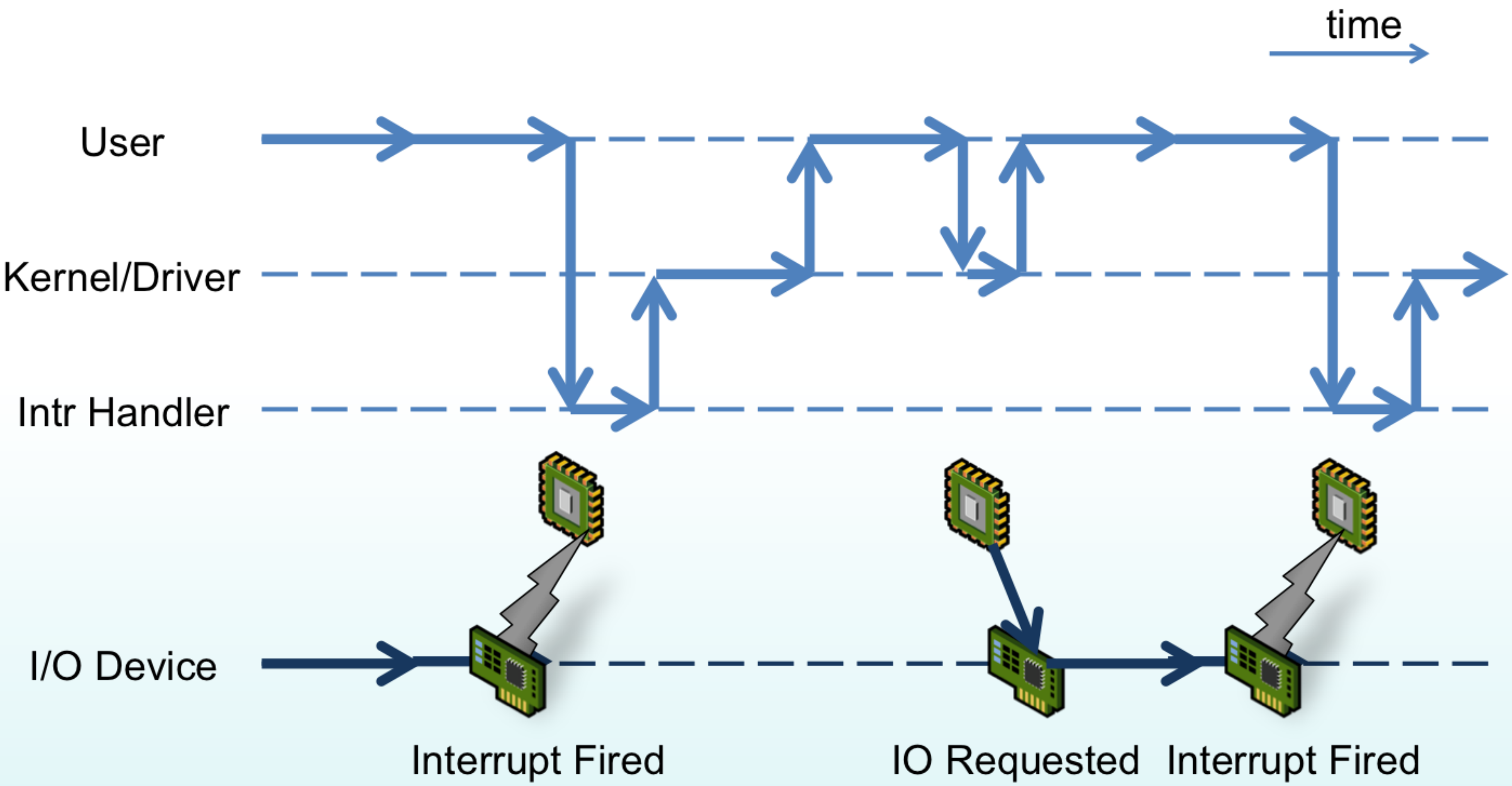
Ultrasonic distance sensor

Interrupts

programs/02_interrupts

The action of an I/O interrupt

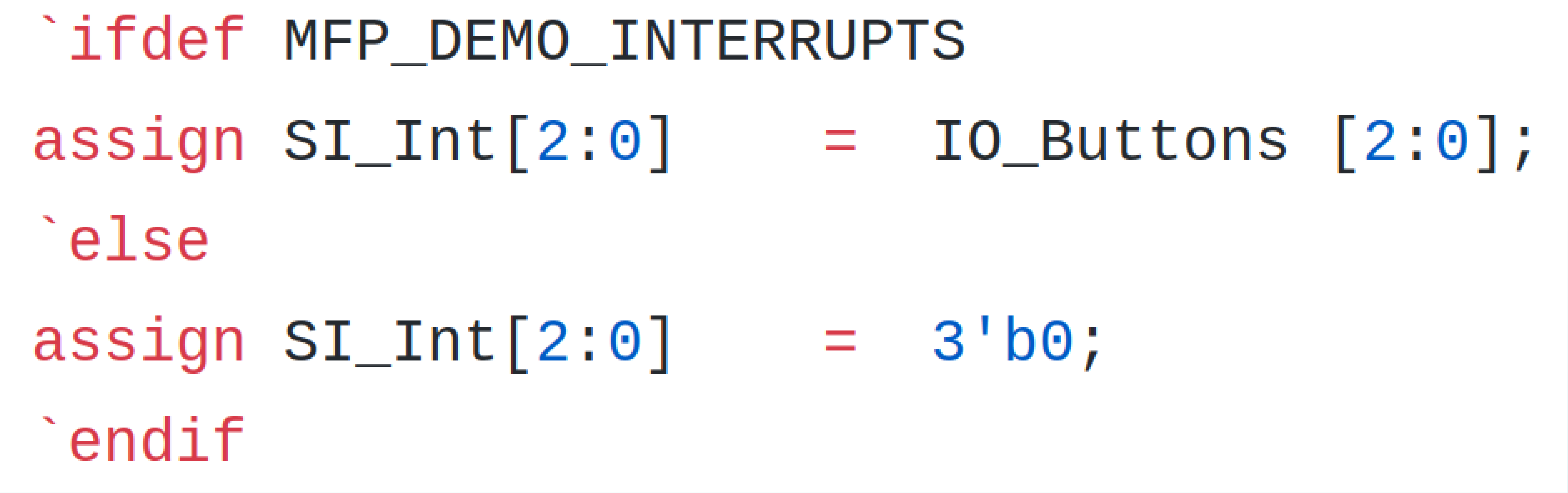
Interrupts in the exercise are driven by pressing buttons on the board



The source of the figure: http://virtualirfan.com/history-of-interrupts

Modifications in hardware

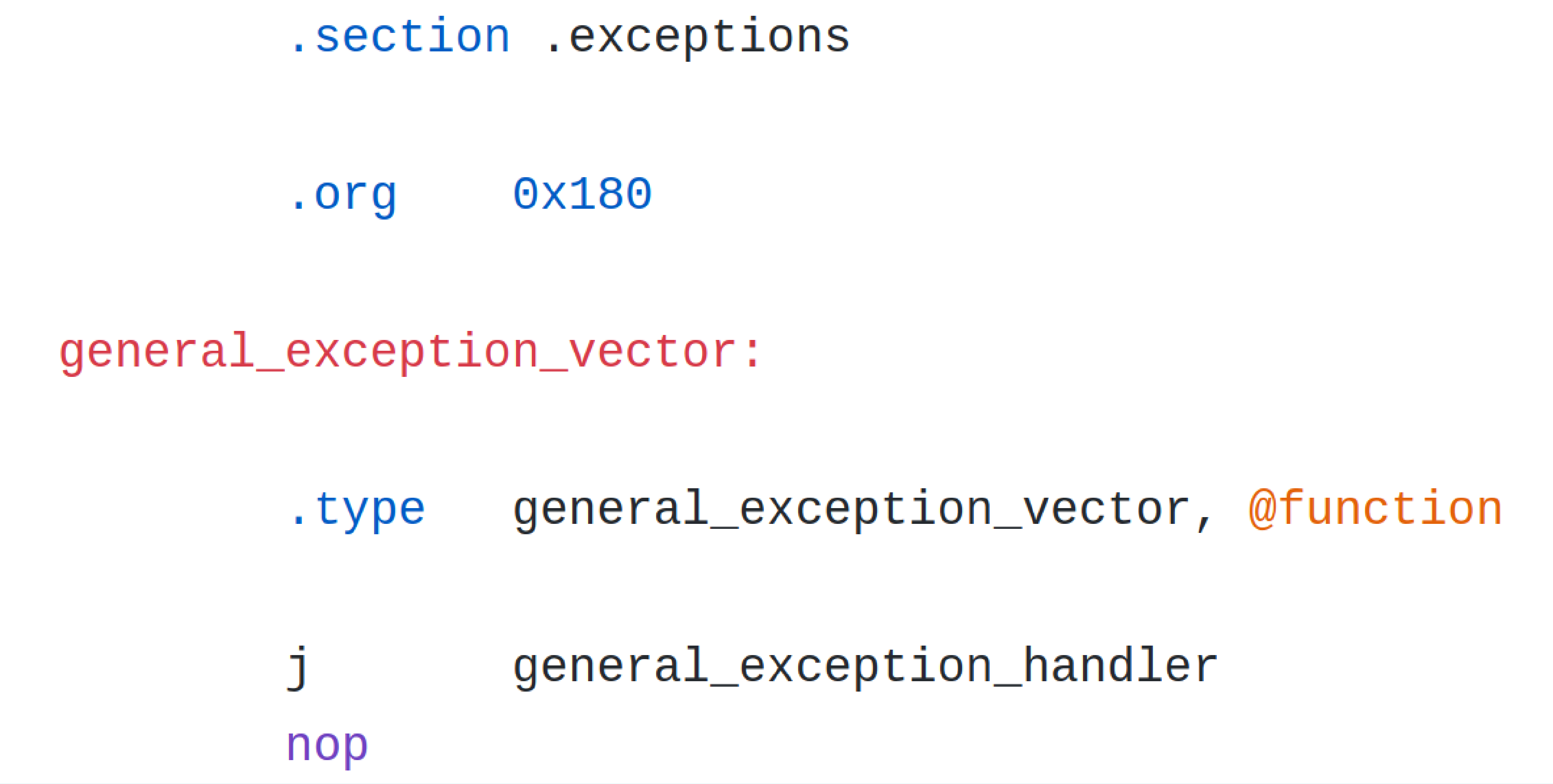
Connecting buttons to interrupt signals inside system_rtl/mfp_system.v



```
`ifdef MFP_DEMO_INTERRUPTS
assign SI_Int[2:0]     =  IO_Buttons [2:0];
`else
assign SI_Int[2:0]     =  3'b0;
`endif
```

Software side

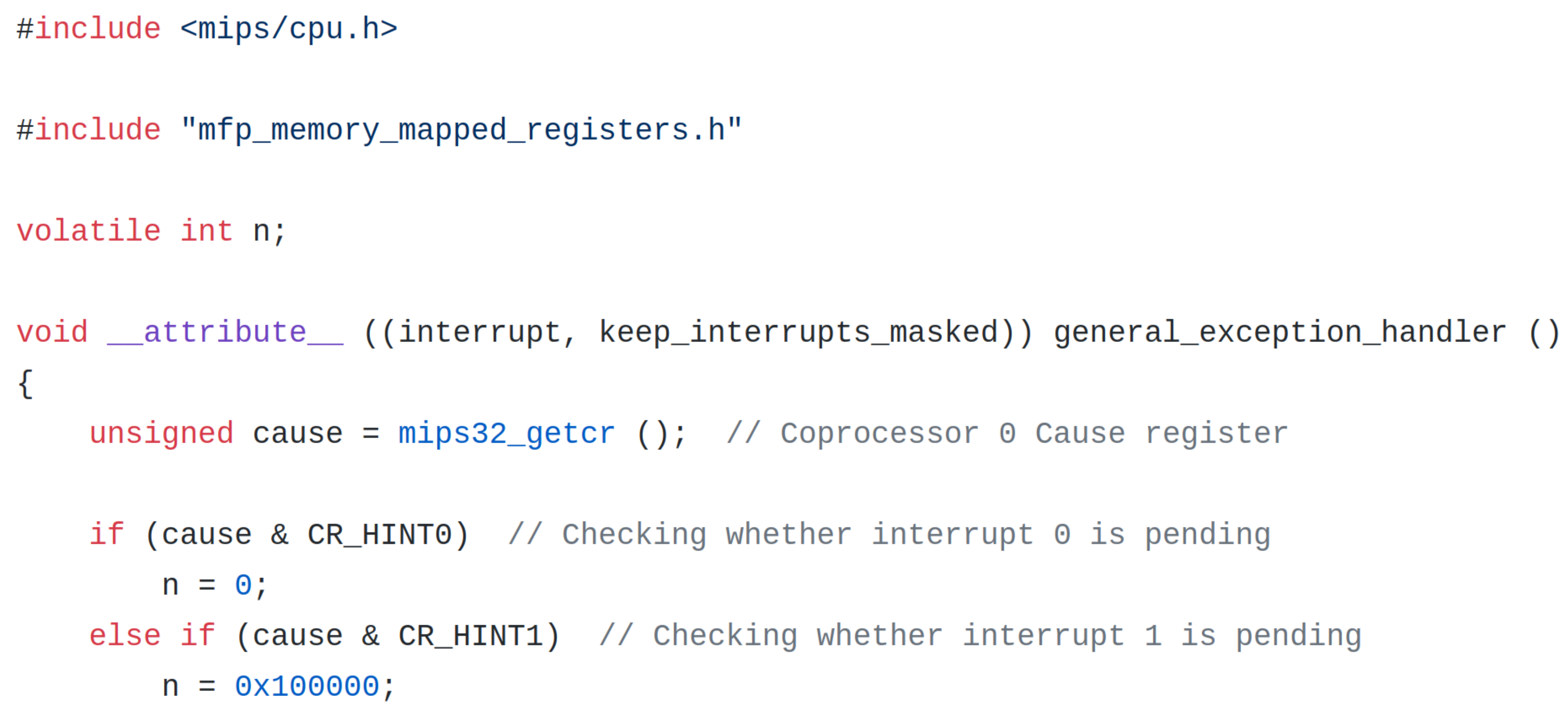
Default general exception handler in programs/02_interrupts/exceptions.S



```
        .section .exceptions

        .org    0x180

general_exception_vector:

        .type   general_exception_vector, @function

        j       general_exception_handler
        nop
```

Software side

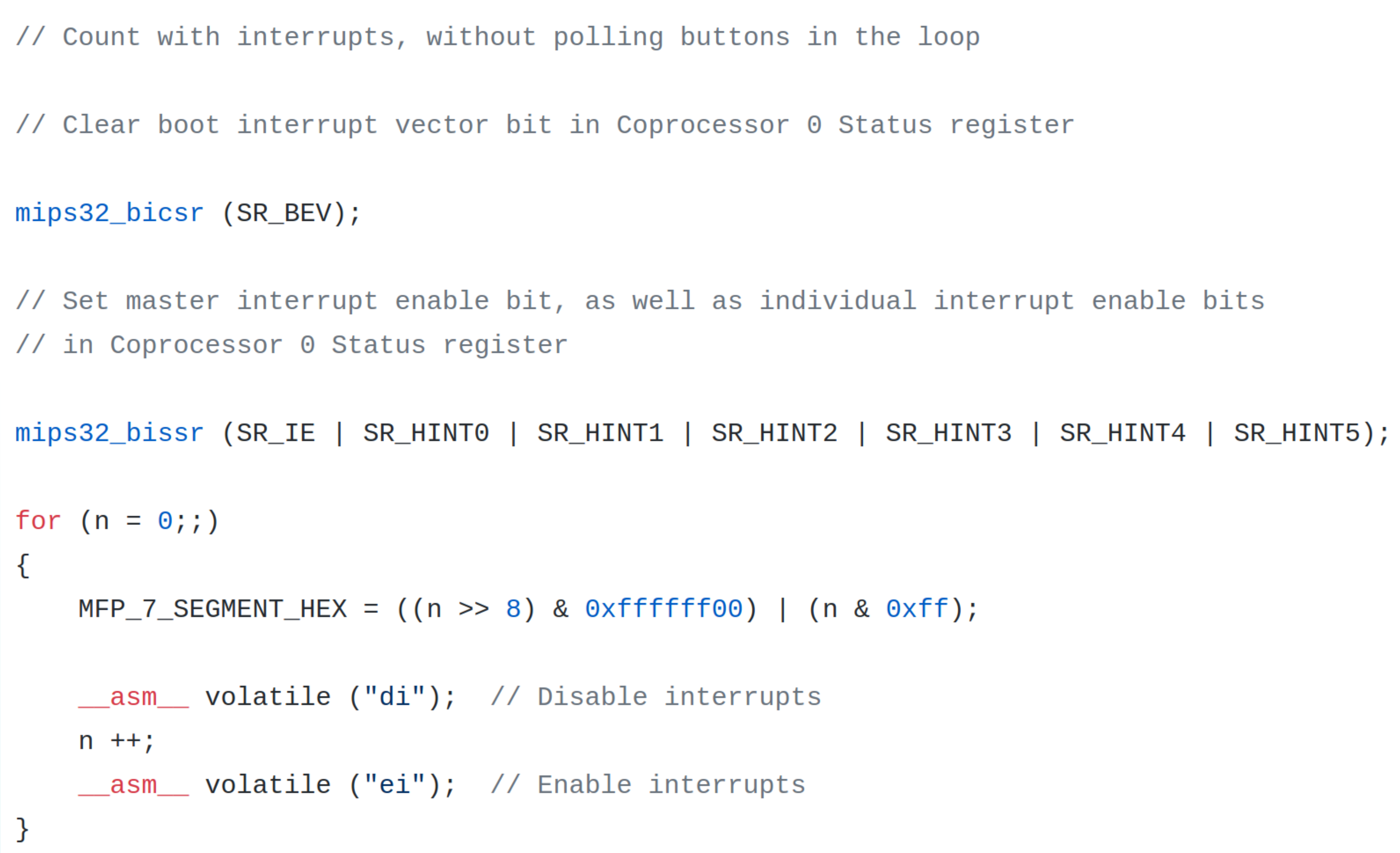
Custom general exception handler in programs/02_interrupts/main.c



```c
#include <mips/cpu.h>

#include "mfp_memory_mapped_registers.h"

volatile int n;

void __attribute__ ((interrupt, keep_interrupts_masked)) general_exception_handler ()
{
    unsigned cause = mips32_getcr ();  // Coprocessor 0 Cause register

    if (cause & CR_HINT0)  // Checking whether interrupt 0 is pending
        n = 0;
    else if (cause & CR_HINT1)  // Checking whether interrupt 1 is pending
        n = 0x100000;
```

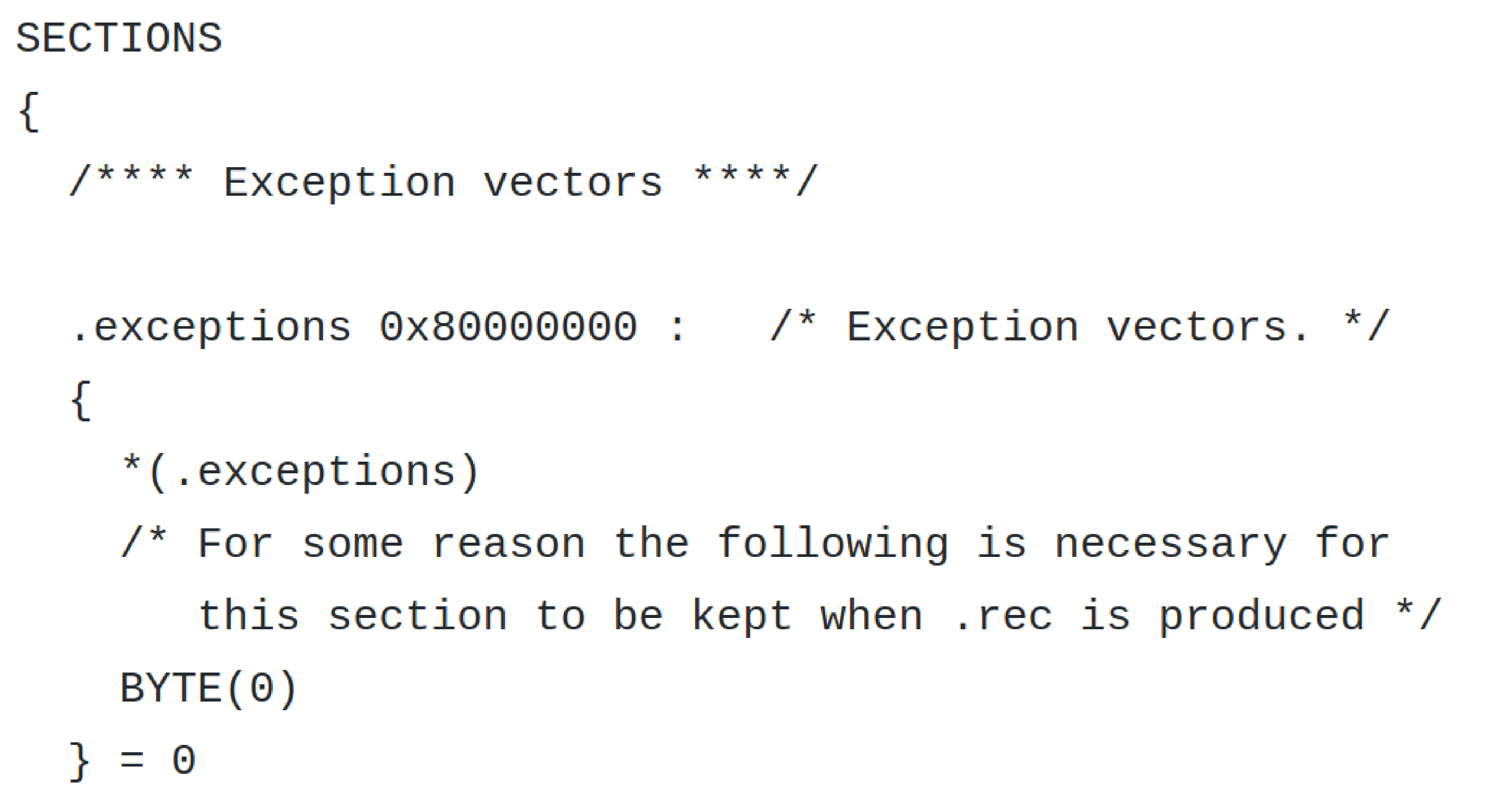
Software side

Setting the interrupts in programs/02_interrupts/main.c



```c
// Count with interrupts, without polling buttons in the loop

// Clear boot interrupt vector bit in Coprocessor 0 Status register

mips32_bicsr (SR_BEV);

// Set master interrupt enable bit, as well as individual interrupt enable bits
// in Coprocessor 0 Status register

mips32_bissr (SR_IE | SR_HINT0 | SR_HINT1 | SR_HINT2 | SR_HINT3 | SR_HINT4 | SR_HINT5);

for (n = 0;;)
{
    MFP_7_SEGMENT_HEX = ((n >> 8) & 0xffffff00) | (n & 0xff);

    __asm__ volatile ("di");  // Disable interrupts
    n ++;
    __asm__ volatile ("ei");  // Enable interrupts
}
```

Changes in linker script in programs/02_interrupts/program.ld



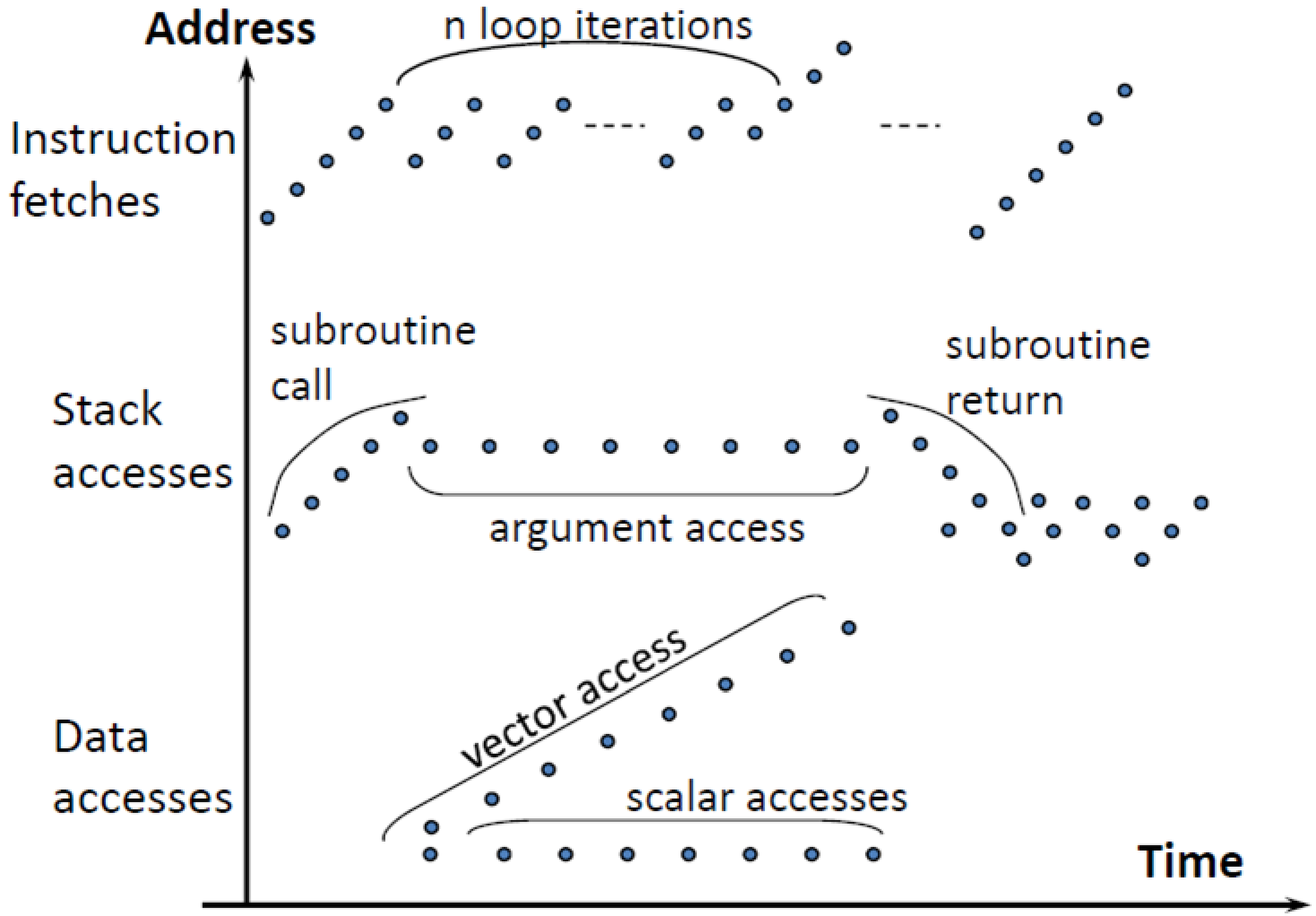
```
SECTIONS
{
  /**** Exception vectors ****/

  .exceptions 0x80000000 :   /* Exception vectors. */
  {
    *(.exceptions)
    /* For some reason the following is necessary for
       this section to be kept when .rec is produced */
    BYTE(0)
  } = 0
```

Observing CPU L1 cache in action

programs/03_cache_misses

Causing different patterns of cache misses

- Caches exploit temporal and spatial locality of instructions and data to improve the processor's performance
- MIPS microAptiv UP core has several cache configurations:
  - I-cache and D-cache
  - 1, 2, 3 or 4 way set associative
  - 1, 2, 4, 8, 16 KB
- MIPS Open Day Package allows to directly observe cache behavior on LED with slow clock feature

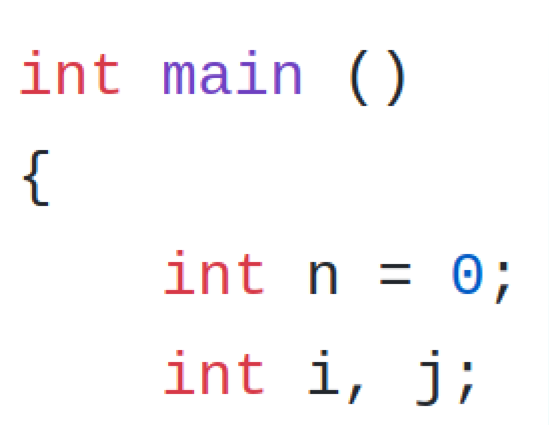


[Memory access patterns from Computer Architecture course by David Wentzlaff from Princeton University. 2011.]

Program with different cache miss patterns

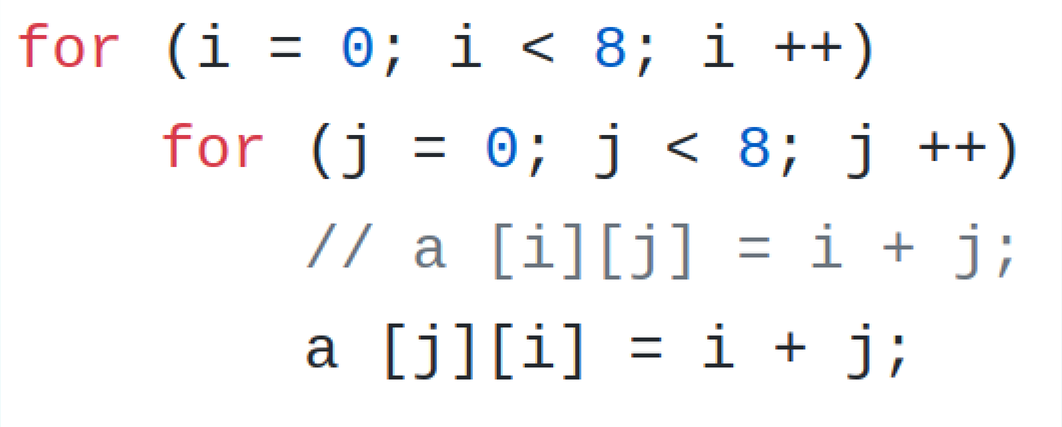
programs/02_cache_misses/main.c

```
int a [8][8];
```



```
int main ()
{
    int n = 0;
    int i, j;
```

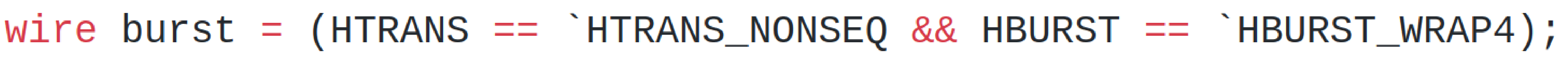
```
// Wait for switch 2

while ((MFP_SWITCHES & 4) == 0)
    ;
```



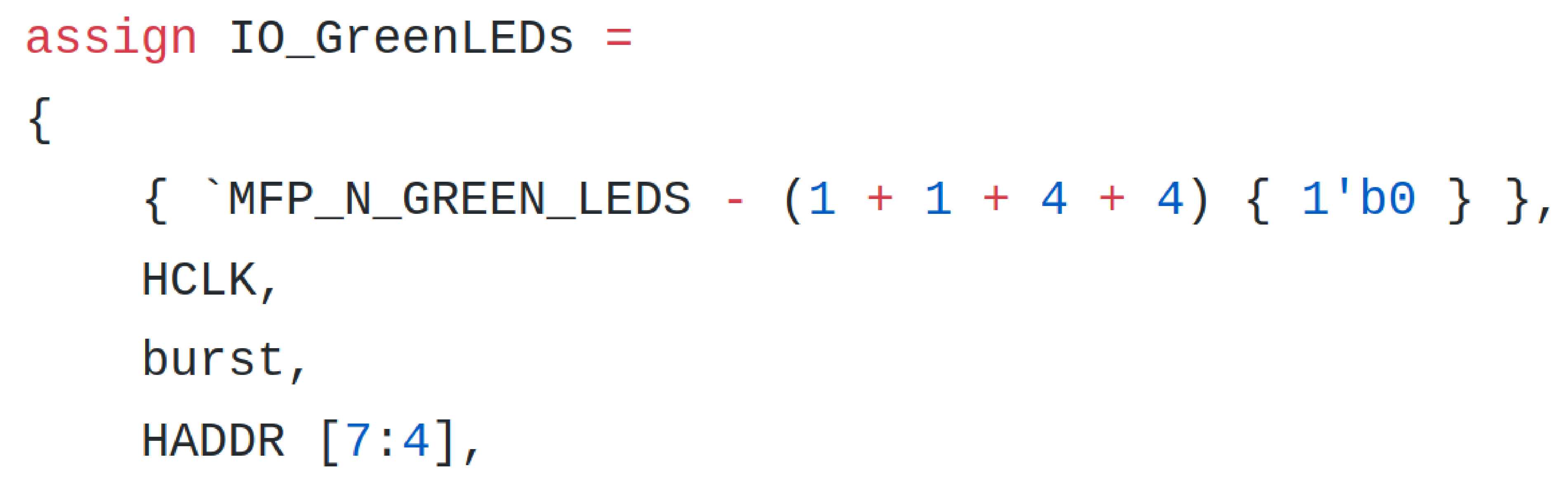
```
for (i = 0; i < 8; i ++)
    for (j = 0; j < 8; j ++)
        // a [i][j] = i + j;
        a [j][i] = i + j;
```

Modifications in hardware

Connecting cache miss signals to LED inside system_rtl/mfp_system.v



```
wire burst = (HTRANS == `HTRANS_NONSEQ && HBURST == `HBURST_WRAP4);
```



```
assign IO_GreenLEDs =
{
    { `MFP_N_GREEN_LEDS - (1 + 1 + 4 + 4) { 1'b0 } },
    HCLK,
    burst,
    HADDR [7:4],
```

Different patterns of cache misses in time

Pattern data:

Miss/Blink, Hit/Nothing, Nothing, Nothing
Miss/Blink, Hit/Nothing, Nothing, Nothing

. . . . . .

	a [0][0]	a [0][1]	a [0][2]	a [0][3]
a [0][0]	0	1	2	3
a [0][4]	4	5	6	7
a [1][0]	8	9	10	11
a [1][4]	12	13	14	15
a [2][0]	16	17	18	19

Pattern for data:

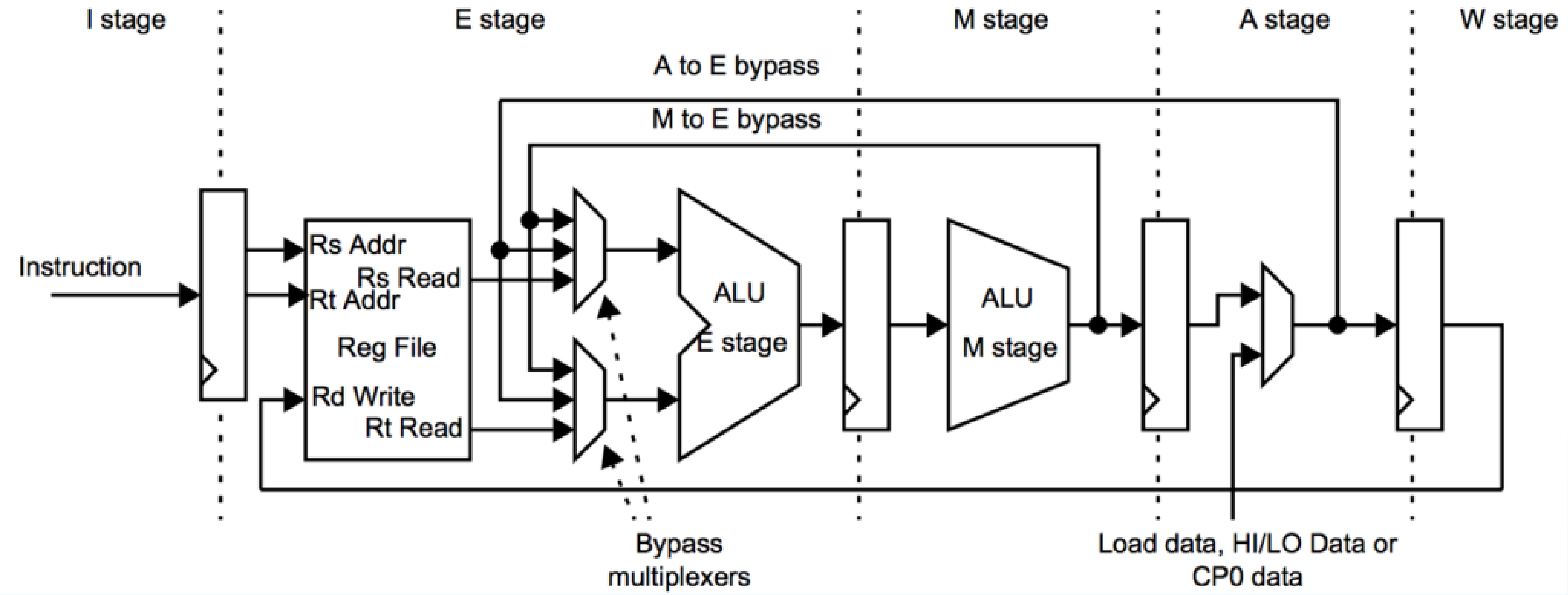
Series of 8 misses, then 24 hits
Series of 8 misses, then 24 hits again
Watch for misses because of instructions

	a [0][0]	a [0][1]	a [0][2]	a [0][3]
a [0][0]	0	8	16	24
a [0][4]	32	40	48	56
a [1][0]	1	9	17	25
a [1][4]	33	41	49	57
a [2][0]	2	10	18	26

Exposing CPU Pipeline Bypasses

programs/04_pipeline_bypasses

Pipeline bypasses and code that exposes them

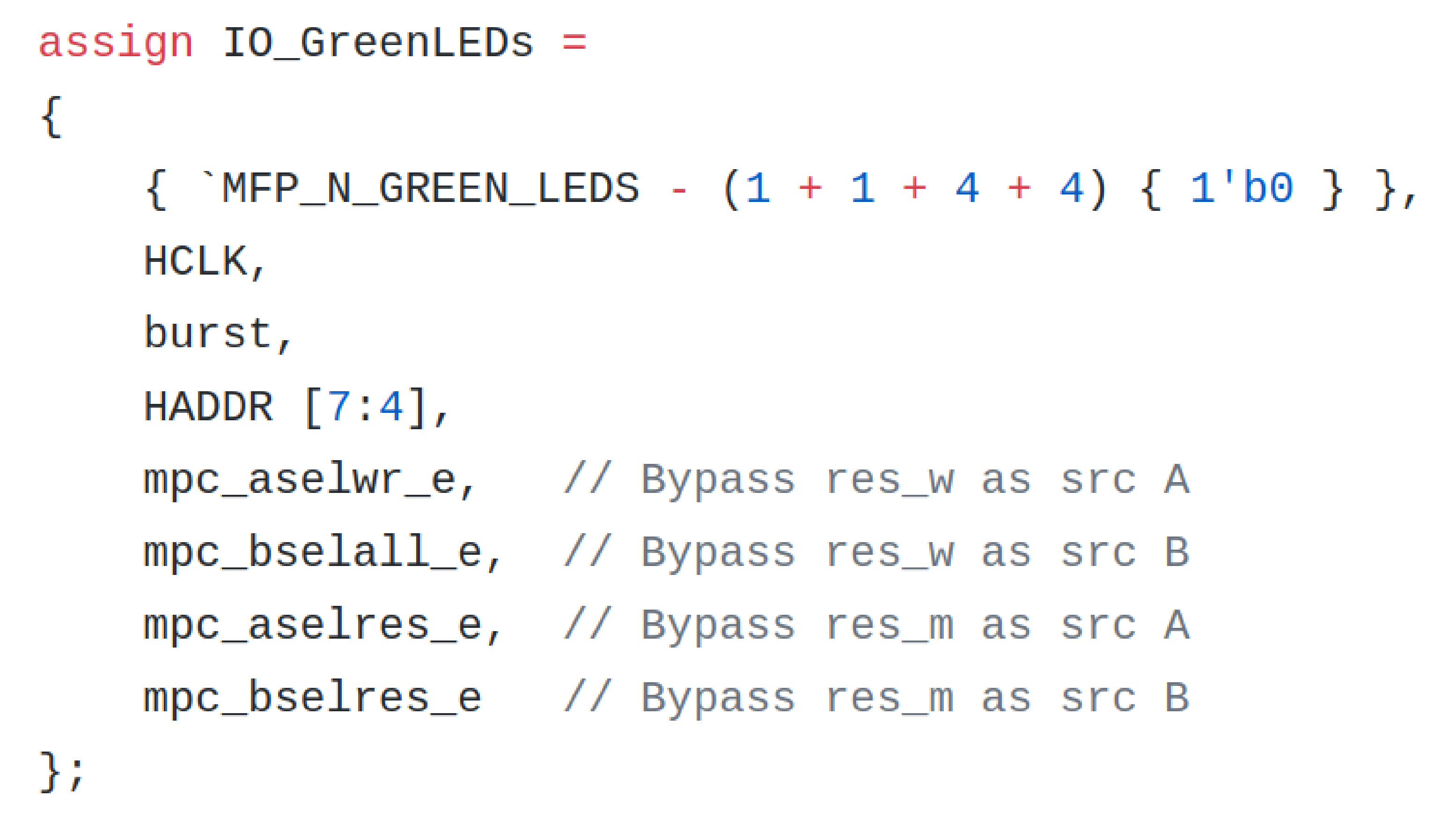


```
for (;;)
{
    __asm__ volatile ("addiu $9,  $8, 1");
    __asm__ volatile ("addiu $10, $9, 1");
```

```
for (;;)
{
    __asm__ volatile ("addiu $9,  $8, 1");
    __asm__ volatile ("addiu $10, $8, 1");
```

Modifications in hardware

Connecting pipeline bypass signals to LEDs inside system_rtl/mfp_system.v

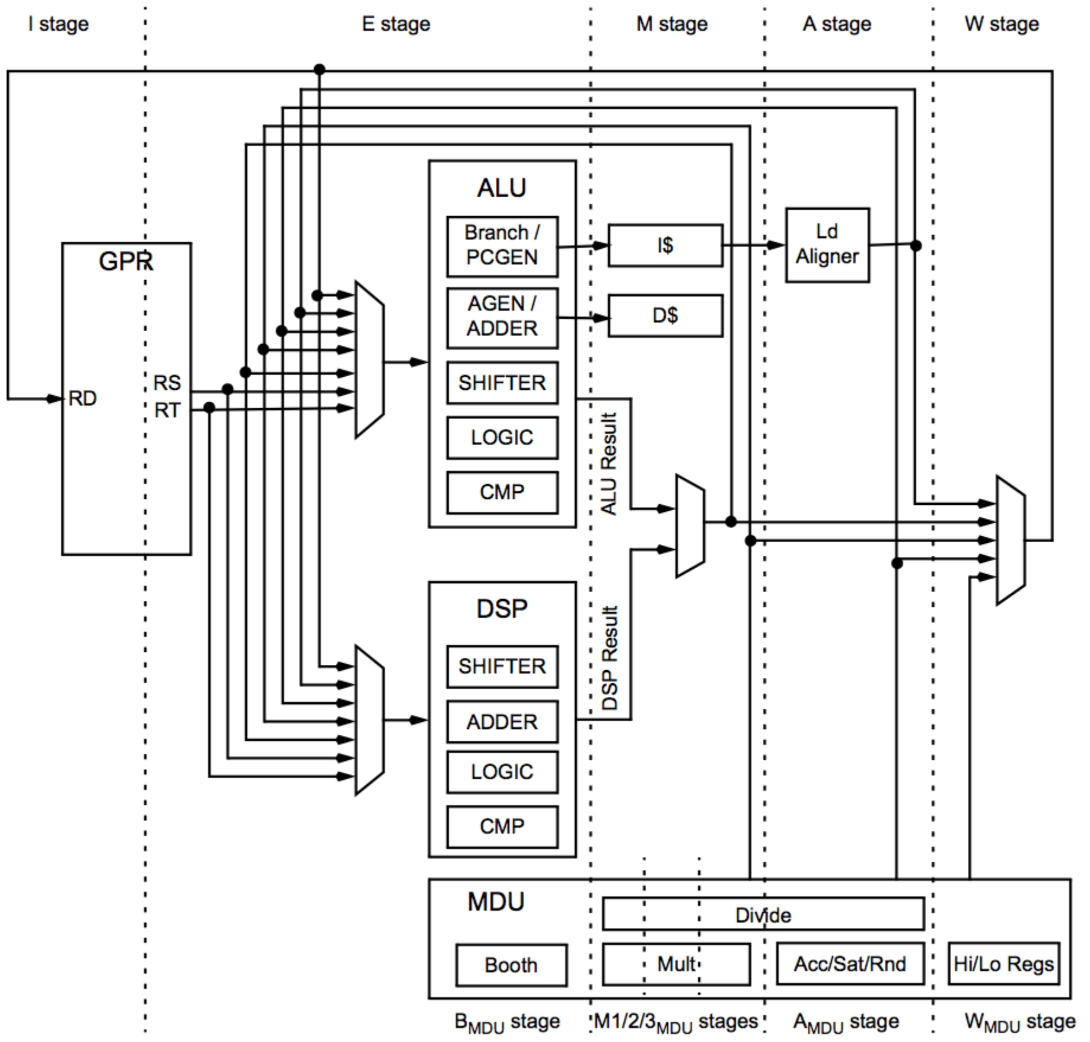


```
assign IO_GreenLEDs =
{
    { `MFP_N_GREEN_LEDS - (1 + 1 + 4 + 4) { 1'b0 } },
    HCLK,
    burst,
    HADDR [7:4],
    mpc_aselwr_e,   // Bypass res_w as src A
    mpc_bselall_e,  // Bypass res_w as src B
    mpc_aselres_e,  // Bypass res_m as src A
    mpc_bselres_e   // Bypass res_m as src B
};
```

More pipelining to explore in microAptiv UP core

- GPR – general purpose registers
- DSP – extension for accelerating Digital Signal Processing algorithms
  - Such as digital filters, FFT
  - Uses light vector operations
  - Options for saturation and rounding
- MDU – Multiply / Divide Unit
  - Different area/performance options
  - Configurable in MIPS Open core
  - Fixed in MIPSfpga

Note: DSP and MDU options are available in full MIPS microAptiv UP core, not in basic configuration of MIPSfpga. User can configure full core and replace core source in MIPSfpga.



MIPS Open™

Take a Break

Build Your own SoC
Present Your Innovation

Build Your own SoC
Present Your Innovation

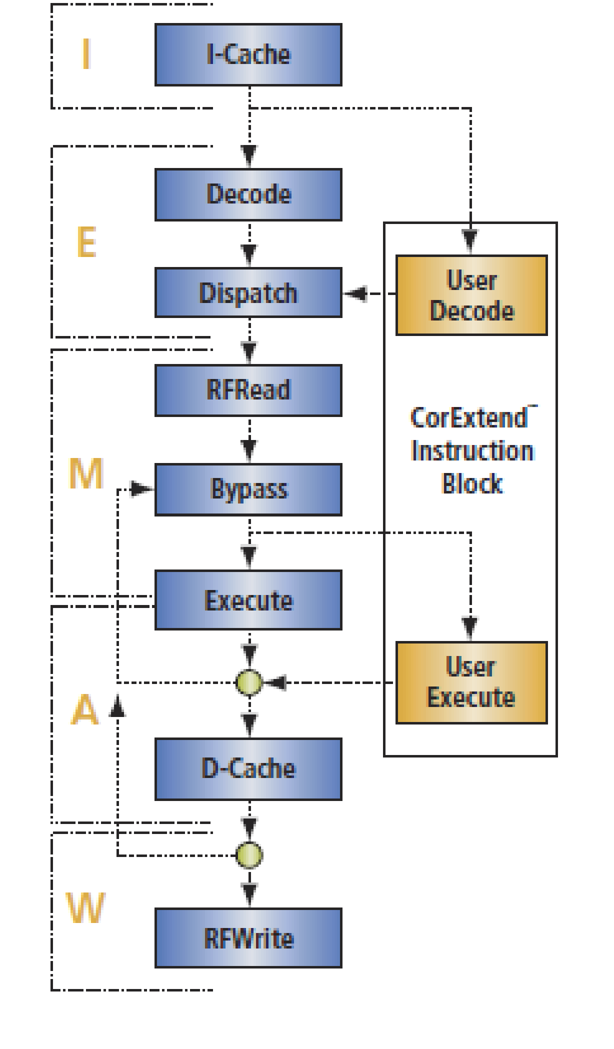
https://www.mipsopen.com/forums/forum/mips-open-developer-day-june-4th-2019/

https://www.mipsopen.com/forums/forum/mips-open-developer-day-june-4th-2019/

Extending CPU with CorExtend interface UDI – User Defined Instructions

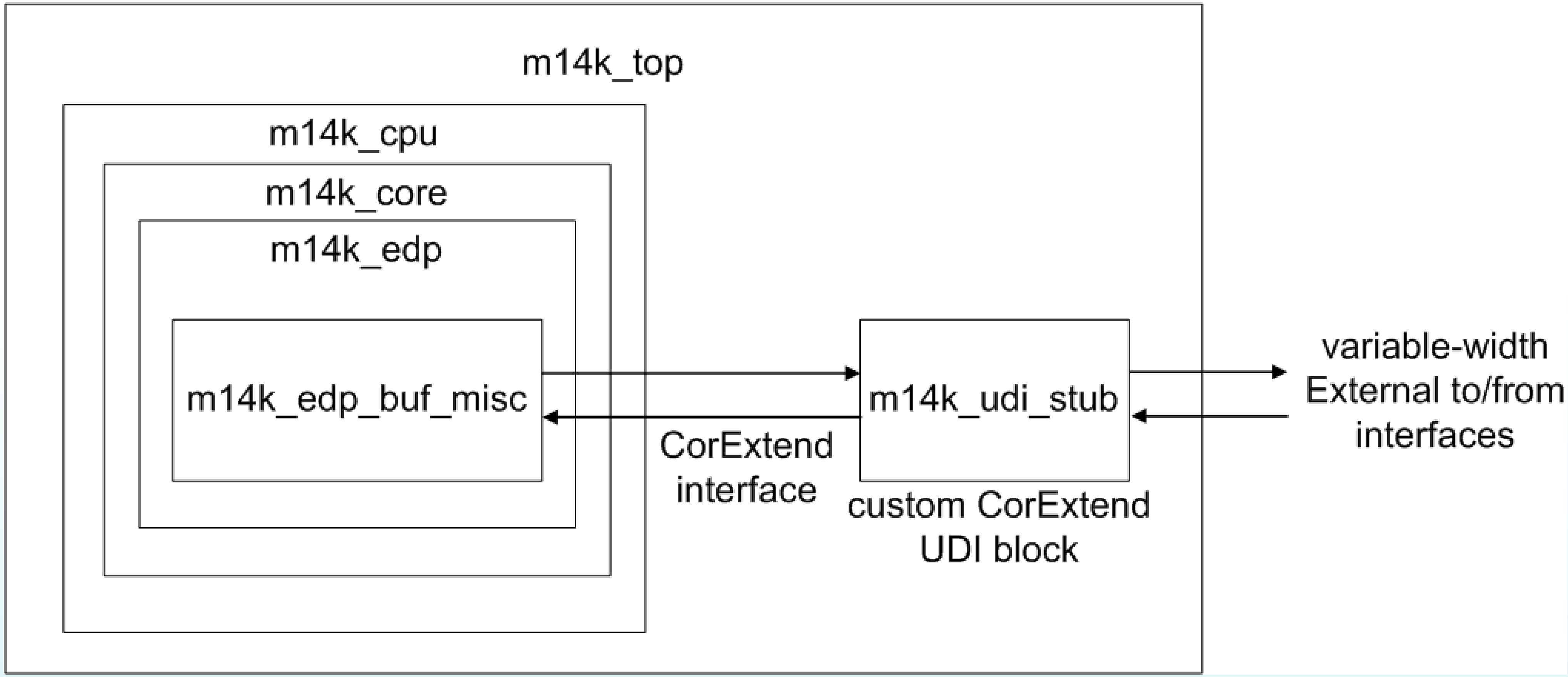
An interface option: CorExtend

- Another name for UDI, User-Defined Instructions.
- Easy to use mechanism for adding new instructions
- User implements a custom block in Verilog.
- Instructions read from two general-purpose registers and write back into a specified register.
- The added instructions do not have to stall the pipe.
- The added instructions can stall the pipe if necessary.
- There is a mechanism to kill the instructions in event of a processor exception.
- The block can have internal state and connect to outside logic.



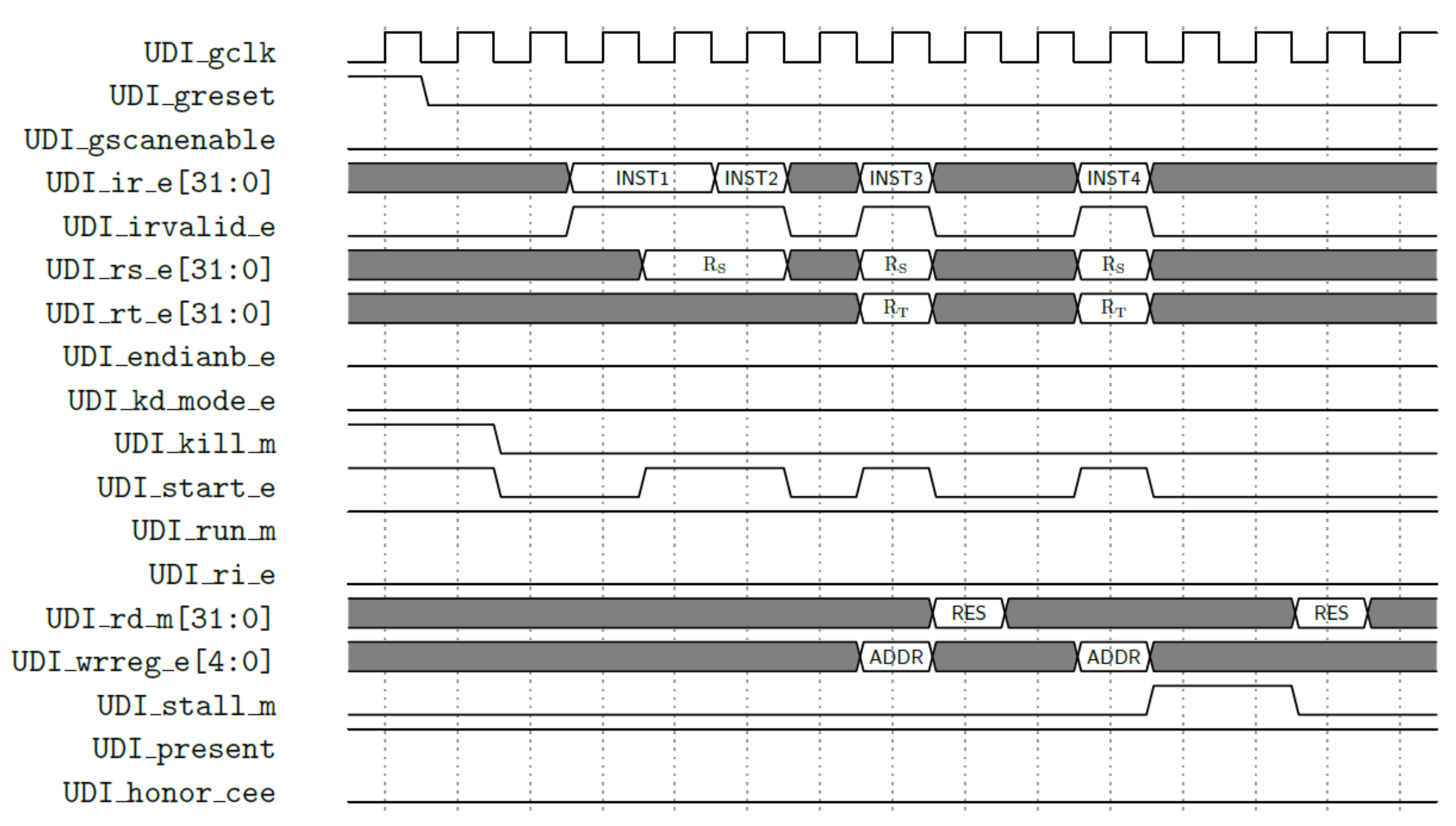
CorExtend block inside the core

http://zatslogic.blogspot.com/2016/01/using-mips-microaptiv-up-processor.html



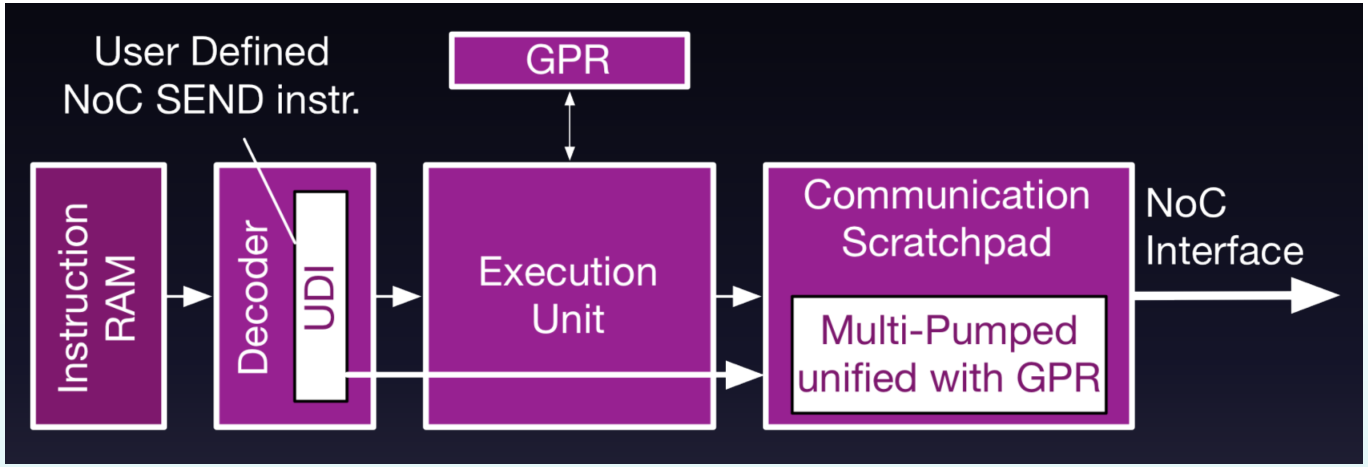
CorExtend instruction processing

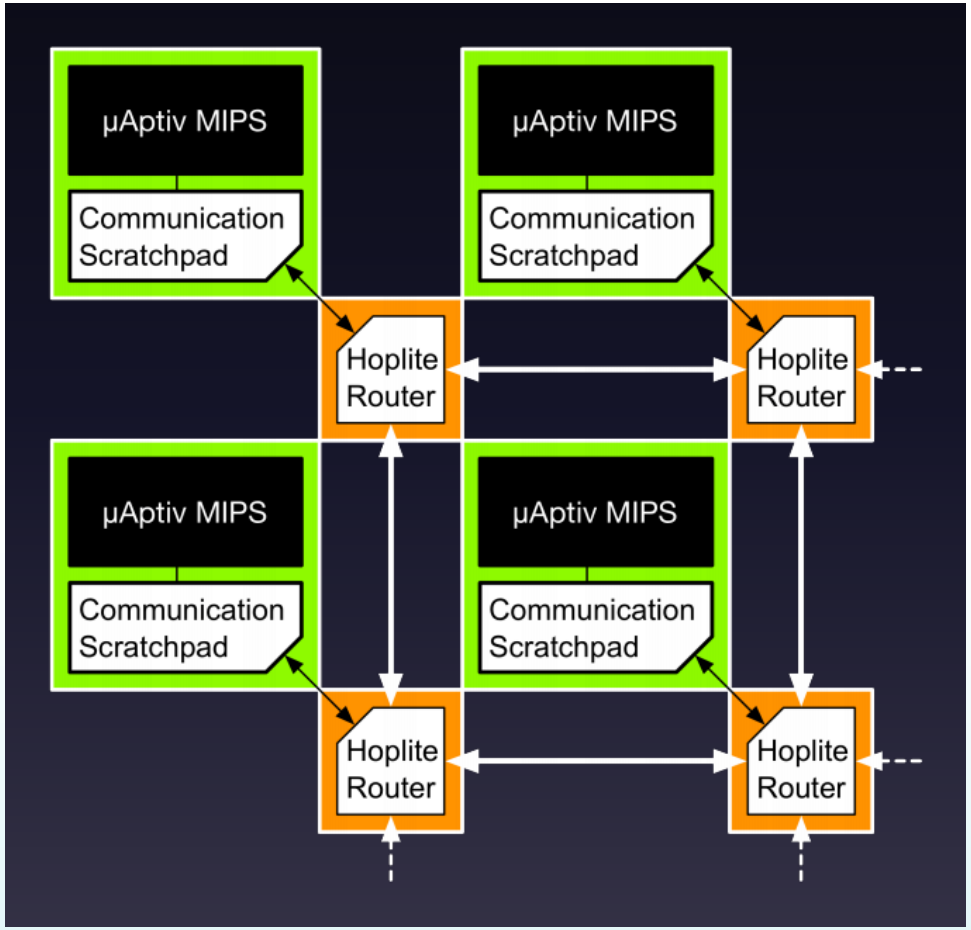
http://zatslogic.blogspot.com/2016/01/using-mips-microaptiv-up-processor.html



An example of UDI use for multiprocessing

- 120 cores working on Terasic DE5-Net board with Altera / Intel FPGA
- The instructions to send messages between the processors in non-coherent mesh





http://www.isfpga.org/fpga2017/slides/D2_S3_04.pdf and http://nachiket.github.io/publications/mips_fpga2017.pdf

WAVE®
COMPUTING

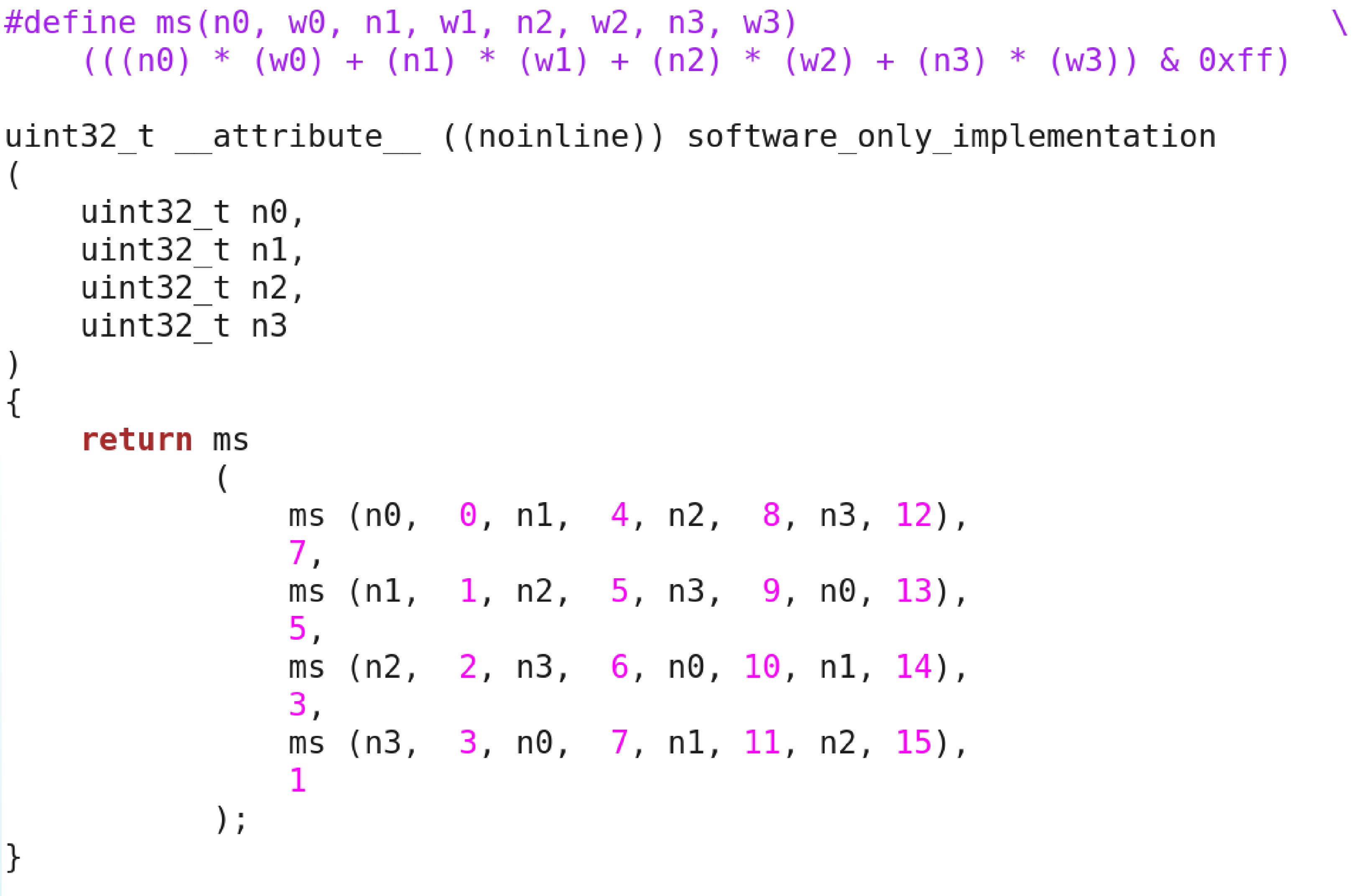
A proof of concept using CorExtend for AI

MIPS Open™

A proof of concept using CorExtend for AI

- Analyze an AI algorithm
- Define a formula to compute in hardware
  - D0 * W0 + D1 * W1 + D2 * W2 + D3 * W3
- Define the format of CorExtend instructions to accelerate the computation
- Implement a custom CorExtend block
- Create a set of C macros for the programming convenience
- Create two implementations of the algorithm
  - Pure software
  - Mixed software-hardware
- Analyze the generated assembly code
- Run the comparison benchmark

A test example: the algorithm in software



```
#define ms(n0, w0, n1, w1, n2, w2, n3, w3)                            \
    (((n0) * (w0) + (n1) * (w1) + (n2) * (w2) + (n3) * (w3)) & 0xff)

uint32_t __attribute__ ((noinline)) software_only_implementation
(
    uint32_t n0,
    uint32_t n1,
    uint32_t n2,
    uint32_t n3
)
{
    return ms
           (
               ms (n0,  0, n1,  4, n2,  8, n3, 12),
               7,
               ms (n1,  1, n2,  5, n3,  9, n0, 13),
               5,
               ms (n2,  2, n3,  6, n0, 10, n1, 14),
               3,
               ms (n3,  3, n0,  7, n1, 11, n2, 15),
               1
           );
}
```

Three formats of UDI instructions

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	rs (optional)	rt (optional)	rd/ud/imm	op xxxxx	UDI opcode 01xxxx
6	5	5	5	5	6

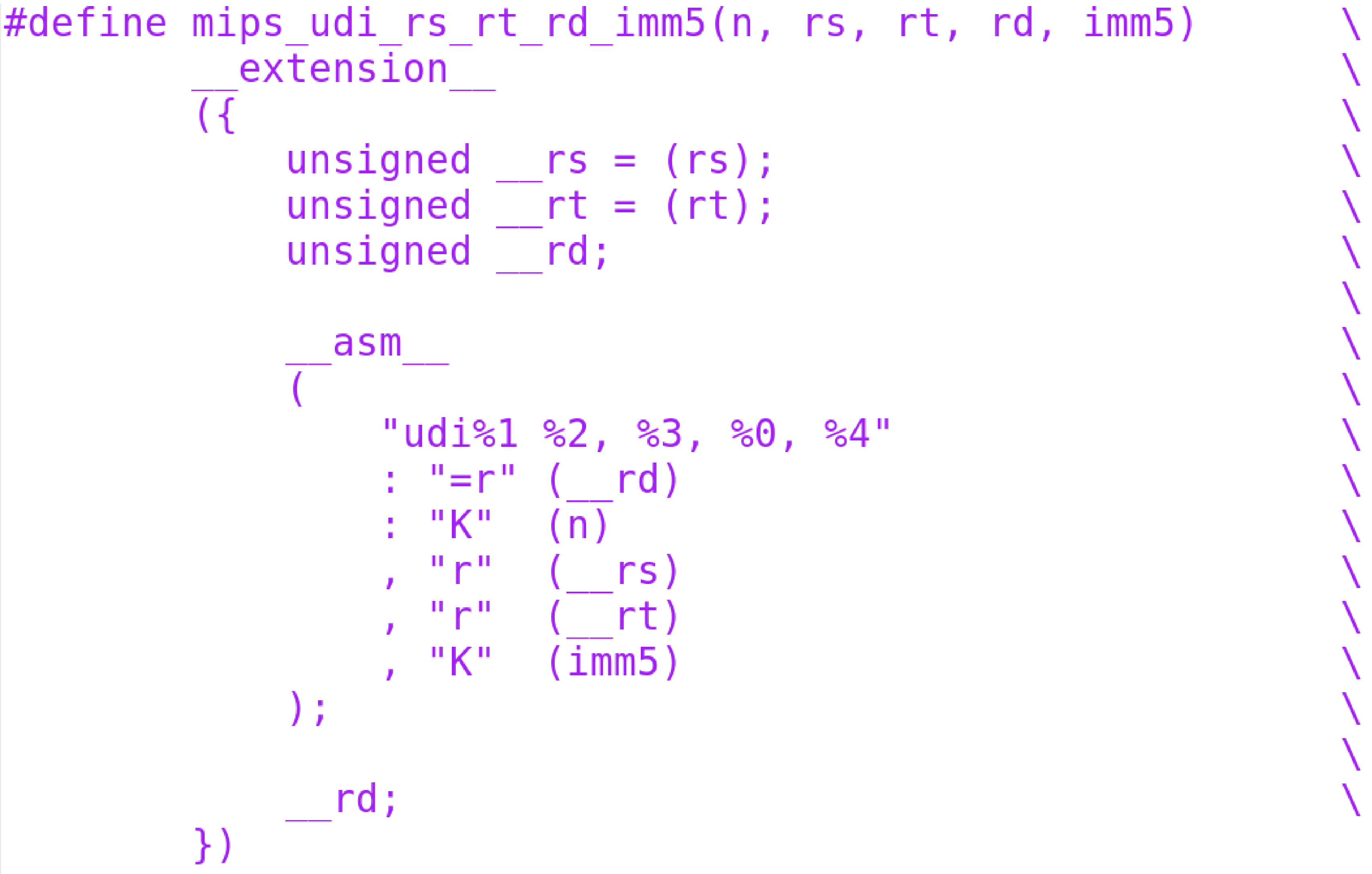
UDIop$_{[3:0]}$ rs, rt, rd, imm5$_{10:6}$	UDI5 $7, $3, $15, 30
UDIop$_{[3:0]}$ rs, rt, imm10$_{15:6}$	UDI0 t0, t1, 665
UDIop$_{[3:0]}$ rs, imm15$_{20:6}$	UDI15 a0, a1, v0, 0x7133

You can make up to 19 bit immediate by reducing the number of UDI instructions

C macro for UDI using GCC __asm__ extension

```
#define mips_udi_rs_rt_rd_imm5_v(n, rs, rt, rd, imm5)    \
        __extension__                                    \
        ({                                               \
            unsigned __rs = (rs);                        \
            unsigned __rt = (rt);                        \
            unsigned __rd;                               \
                                                         \
            __asm__ __volatile__                         \
            (                                            \
                "udi%1 %2, %3, %0, %4"                   \
                : "=r" (__rd)                            \
                : "K"  (n)                               \
                , "r"  (__rs)                            \
                , "r"  (__rt)                            \
                , "K"  (imm5)                            \
            );                                           \
                                                         \
            __rd;                                        \
        })
```

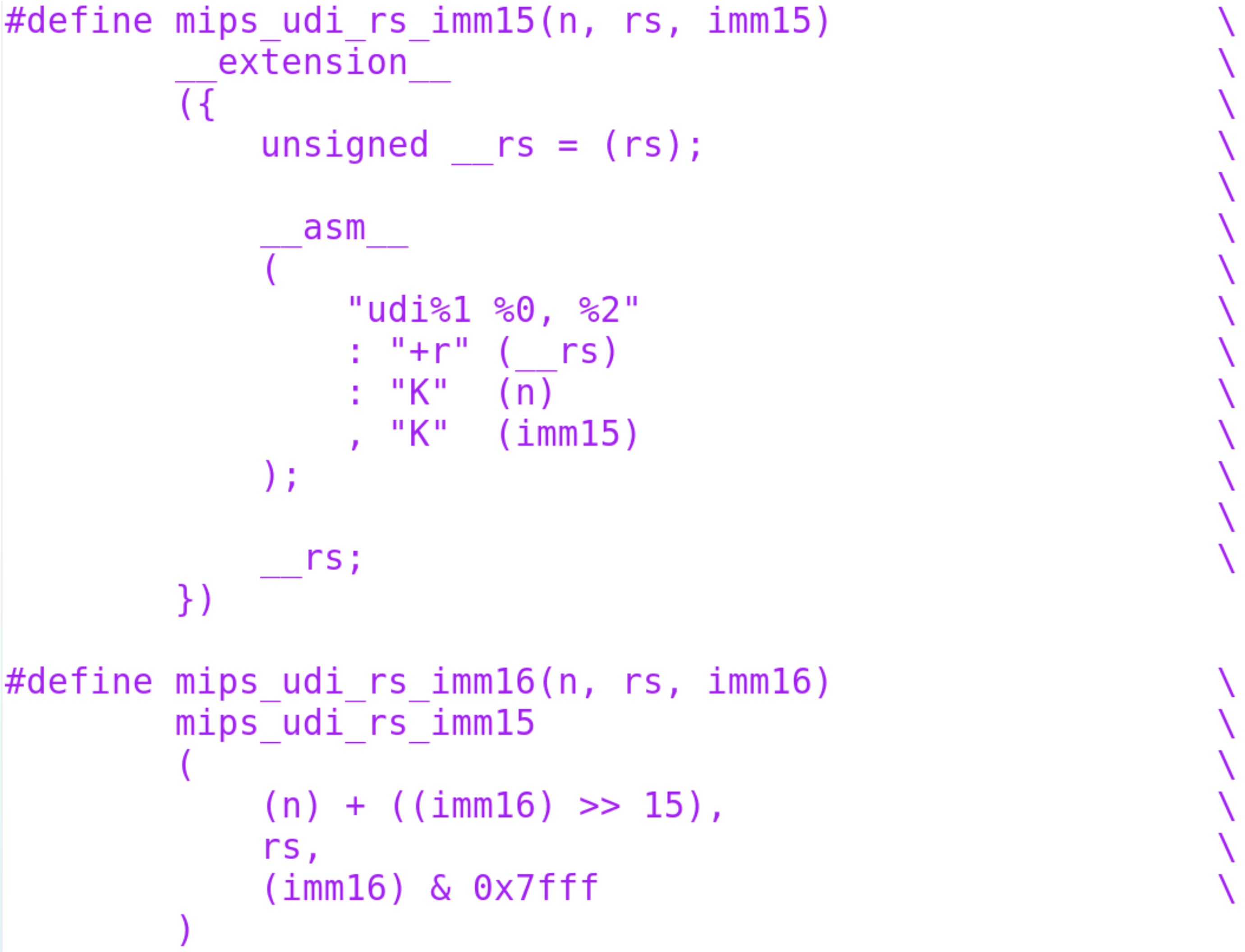
Version without __volatile__ to use additional GCC optimizations



```
#define mips_udi_rs_rt_rd_imm5(n, rs, rt, rd, imm5)       \
        __extension__                                     \
        ({                                                \
            unsigned __rs = (rs);                         \
            unsigned __rt = (rt);                         \
            unsigned __rd;                                \
                                                          \
            __asm__                                       \
            (                                             \
                "udi%1 %2, %3, %0, %4"                    \
                : "=r" (__rd)                             \
                : "K"  (n)                                \
                , "r"  (__rs)                             \
                , "r"  (__rt)                             \
                , "K"  (imm5)                             \
            );                                            \
                                                          \
            __rd;                                         \
        })
```

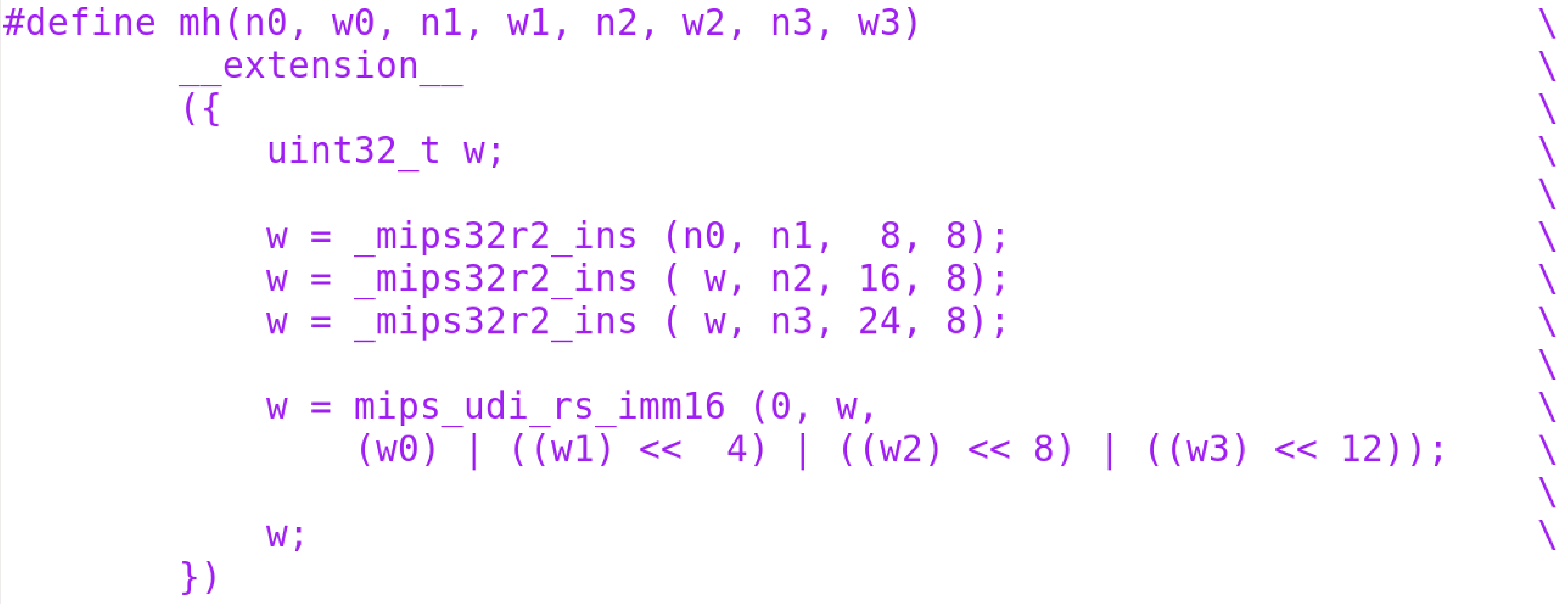
Three formats of UDI instructions

- A format with one register
- 16-bit immediate
- The register is both source and destination
- Number of UDI instructions is reduced to 8
- But we have extra bit for immediate



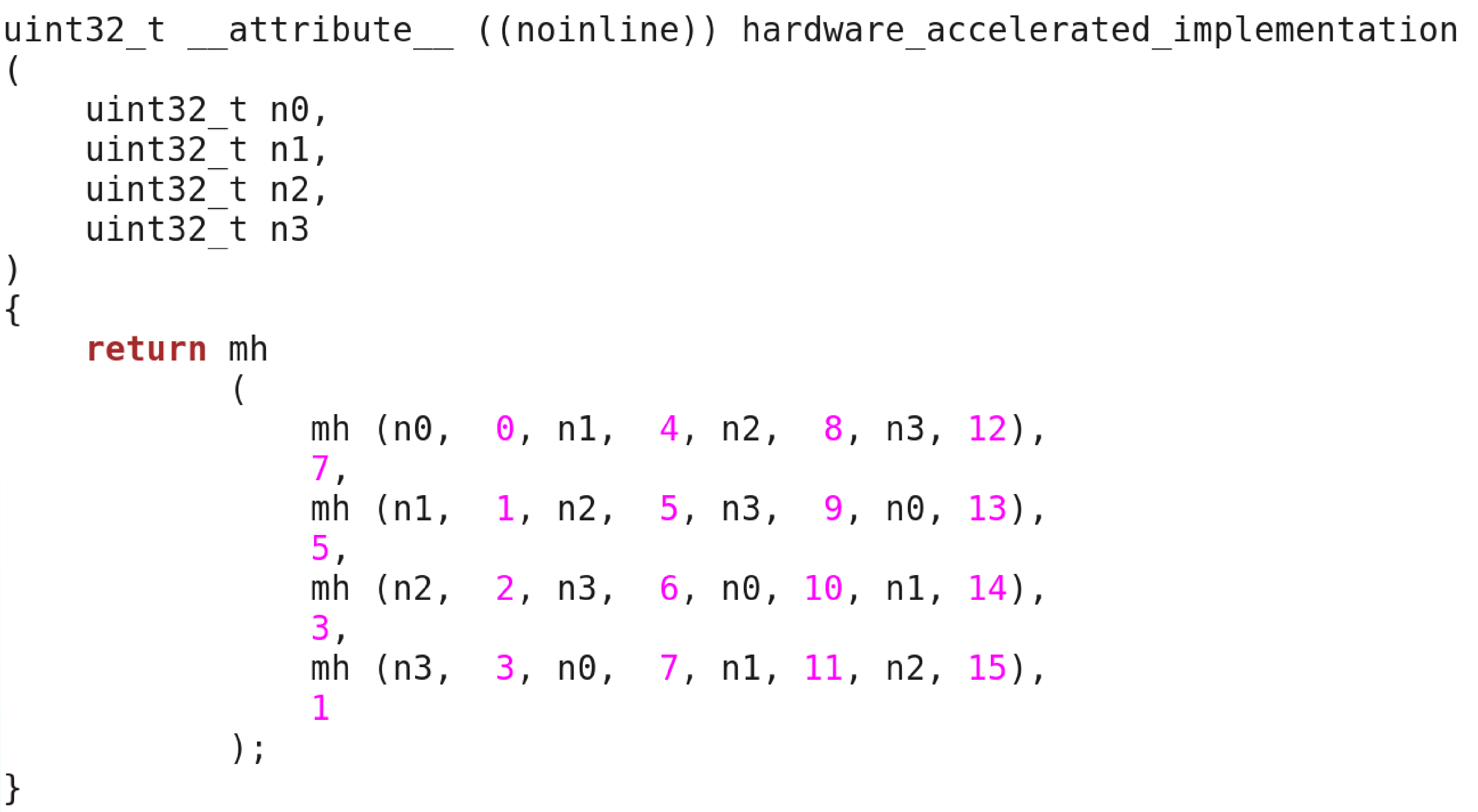
```
#define mips_udi_rs_imm15(n, rs, imm15)                  \
        __extension__                                    \
        ({                                               \
            unsigned __rs = (rs);                        \
                                                         \
            __asm__                                      \
            (                                            \
                "udi%1 %0, %2"                           \
                : "+r" (__rs)                            \
                : "K"  (n)                               \
                , "K"  (imm15)                           \
            );                                           \
                                                         \
            __rs;                                        \
        })

#define mips_udi_rs_imm16(n, rs, imm16)                  \
        mips_udi_rs_imm15                                \
        (                                                \
            (n) + ((imm16) >> 15),                       \
            rs,                                          \
            (imm16) & 0x7fff                             \
        )
```

Algorithm-specific macro uses generic UDI macro



```
#define mh(n0, w0, n1, w1, n2, w2, n3, w3)                             \
        __extension__                                                 \
        ({                                                            \
            uint32_t w;                                               \
                                                                      \
            w = _mips32r2_ins (n0, n1,  8, 8);                        \
            w = _mips32r2_ins ( w, n2, 16, 8);                        \
            w = _mips32r2_ins ( w, n3, 24, 8);                        \
                                                                      \
            w = mips_udi_rs_imm16 (0, w,                              \
                (w0) | ((w1) <<  4) | ((w2) << 8) | ((w3) << 12));    \
                                                                      \
            w;                                                        \
        })
```

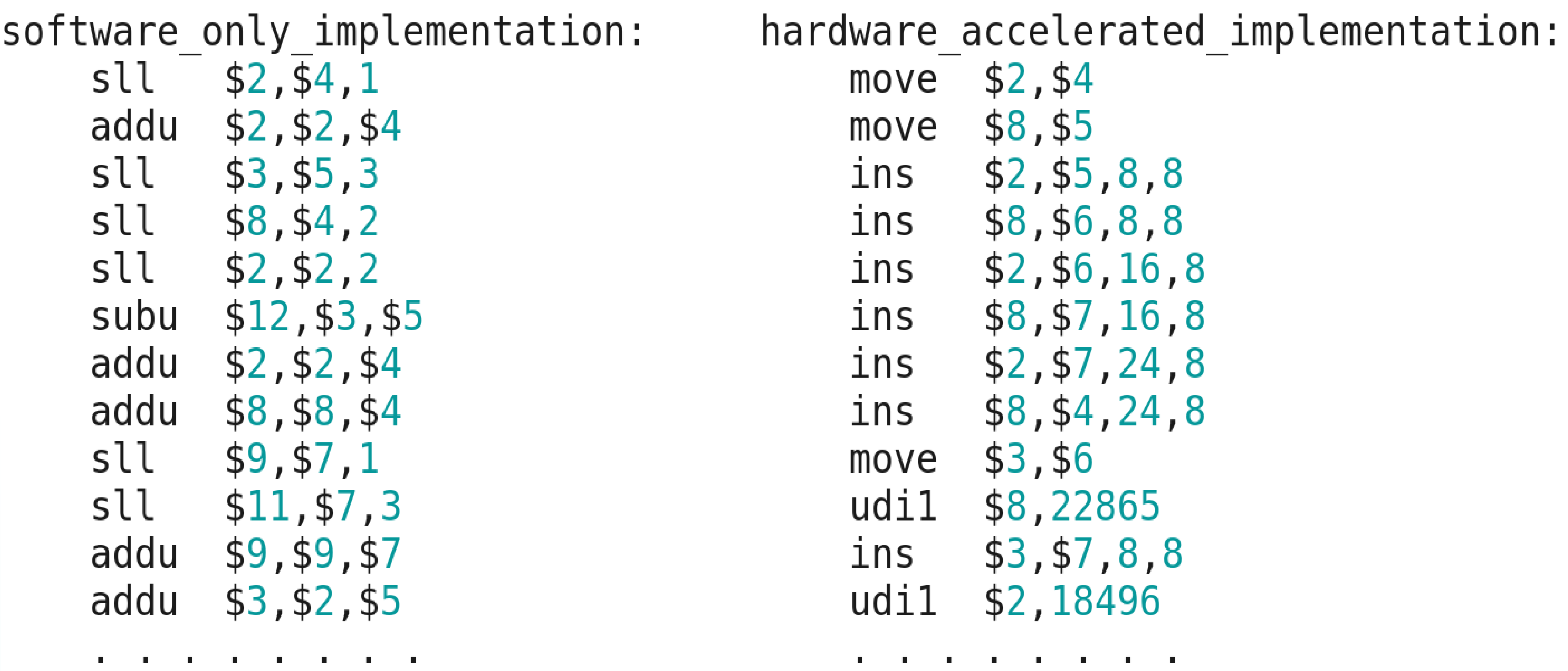
The code in mixed software-hardware is the same



```
uint32_t __attribute__ ((noinline)) hardware_accelerated_implementation
(
    uint32_t n0,
    uint32_t n1,
    uint32_t n2,
    uint32_t n3
)
{
    return mh
           (
               mh (n0,  0, n1,  4, n2,  8, n3, 12),
               7,
               mh (n1,  1, n2,  5, n3,  9, n0, 13),
               5,
               mh (n2,  2, n3,  6, n0, 10, n1, 14),
               3,
               mh (n3,  3, n0,  7, n1, 11, n2, 15),
               1
           );
}
```

Wave Computing © 2019

The generated assembly



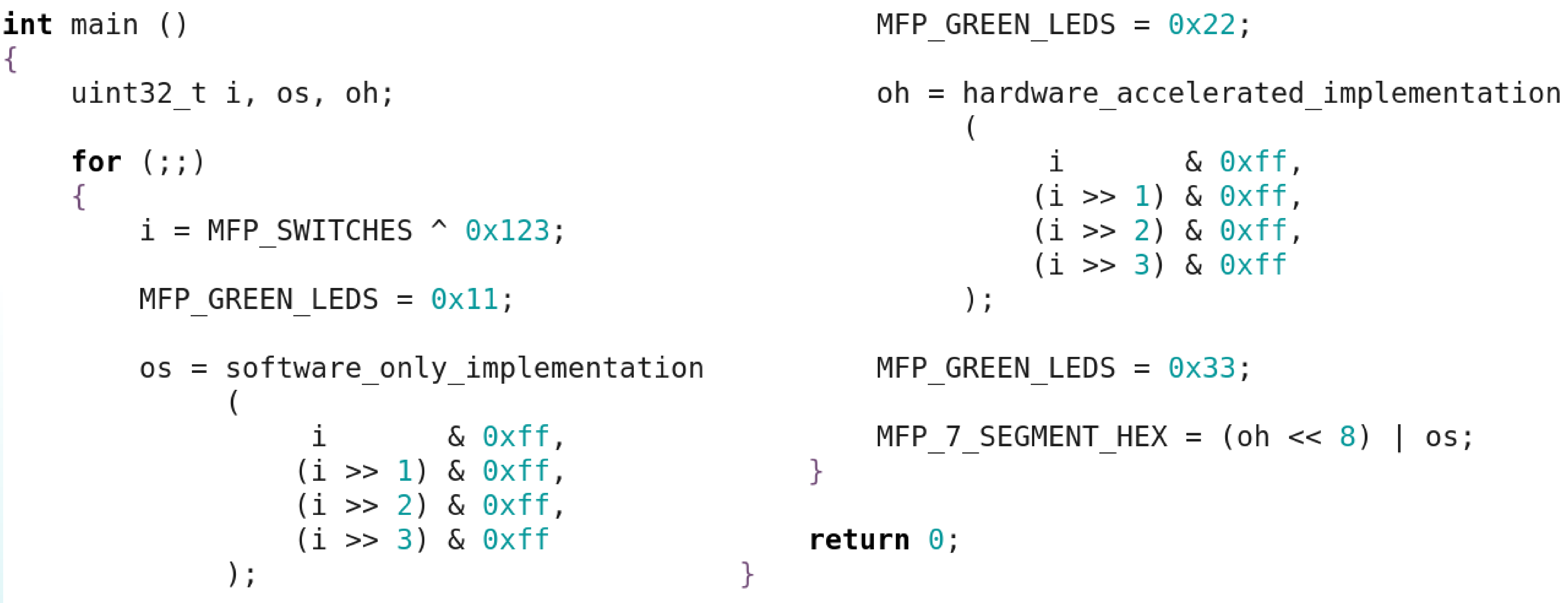
```
software_only_implementation:
        sll     $2,$4,1
        addu    $2,$2,$4
        sll     $3,$5,3
        sll     $8,$4,2
        sll     $2,$2,2
        subu    $12,$3,$5
        addu    $2,$2,$4
        addu    $8,$8,$4
        sll     $9,$7,1
        sll     $11,$7,3
        addu    $9,$9,$7
        addu    $3,$2,$5
        . . . . . . . .
```

```
hardware_accelerated_implementation:
        move    $2,$4
        move    $8,$5
        ins     $2,$5,8,8
        ins     $8,$6,8,8
        ins     $2,$6,16,8
        ins     $8,$7,16,8
        ins     $2,$7,24,8
        ins     $8,$4,24,8
        move    $3,$6
        udi1    $8,22865
        ins     $3,$7,8,8
        udi1    $2,18496
        . . . . . . . .
```

47 instructions without UDI

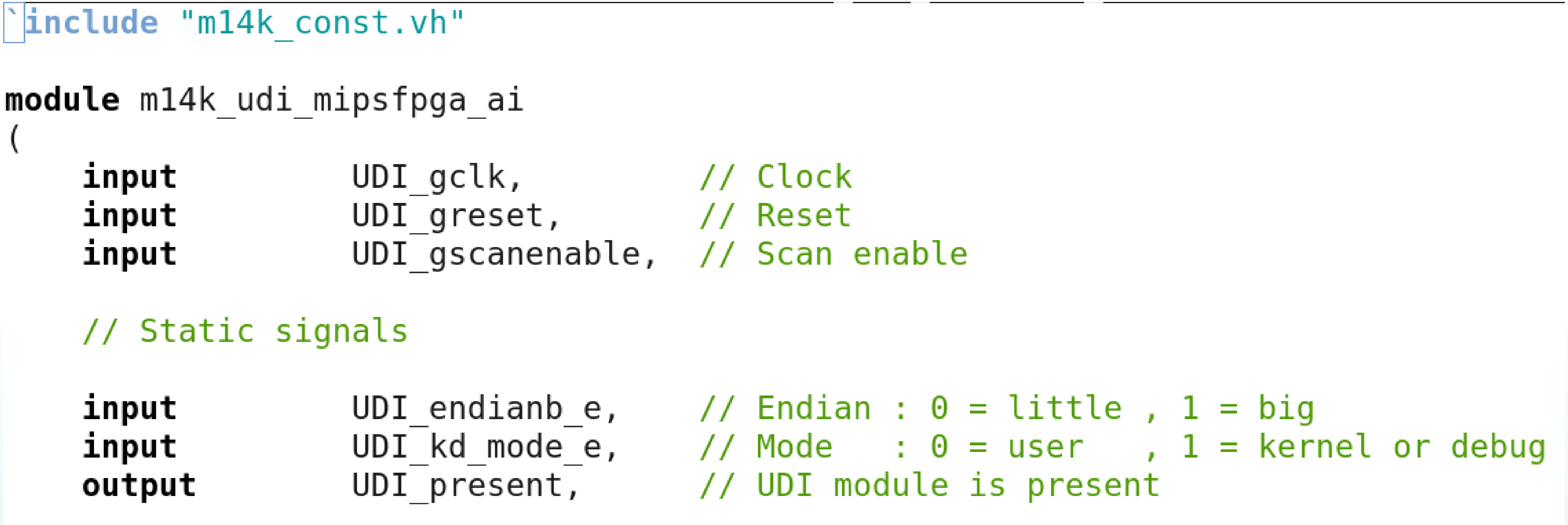
25 instructions with UDI

The software test - uses FPGA board peripherals



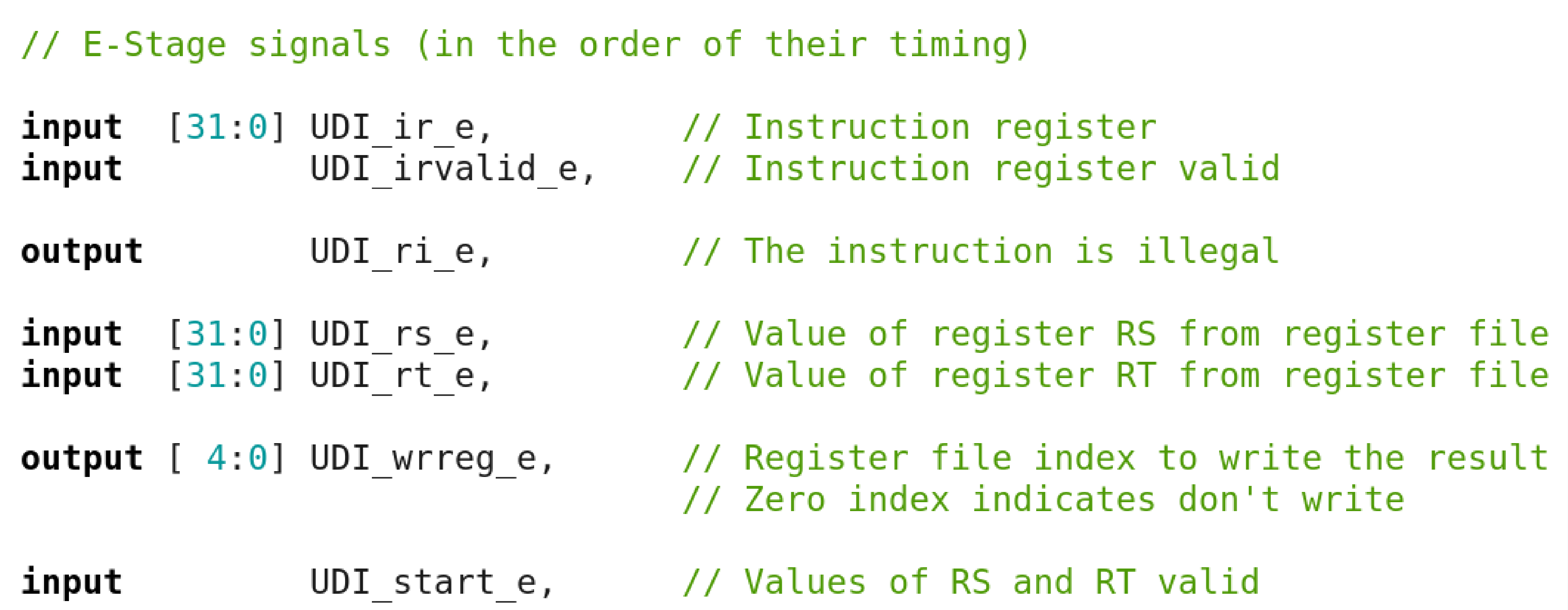
```c
int main ()
{
    uint32_t i, os, oh;

    for (;;)
    {
        i = MFP_SWITCHES ^ 0x123;

        MFP_GREEN_LEDS = 0x11;

        os = software_only_implementation
             (
                  i        & 0xff,
                 (i >> 1) & 0xff,
                 (i >> 2) & 0xff,
                 (i >> 3) & 0xff
             );

        MFP_GREEN_LEDS = 0x22;

        oh = hardware_accelerated_implementation
             (
                  i        & 0xff,
                 (i >> 1) & 0xff,
                 (i >> 2) & 0xff,
                 (i >> 3) & 0xff
             );

        MFP_GREEN_LEDS = 0x33;

        MFP_7_SEGMENT_HEX = (oh << 8) | os;
    }

    return 0;
}
```

CorExtend block code - 1



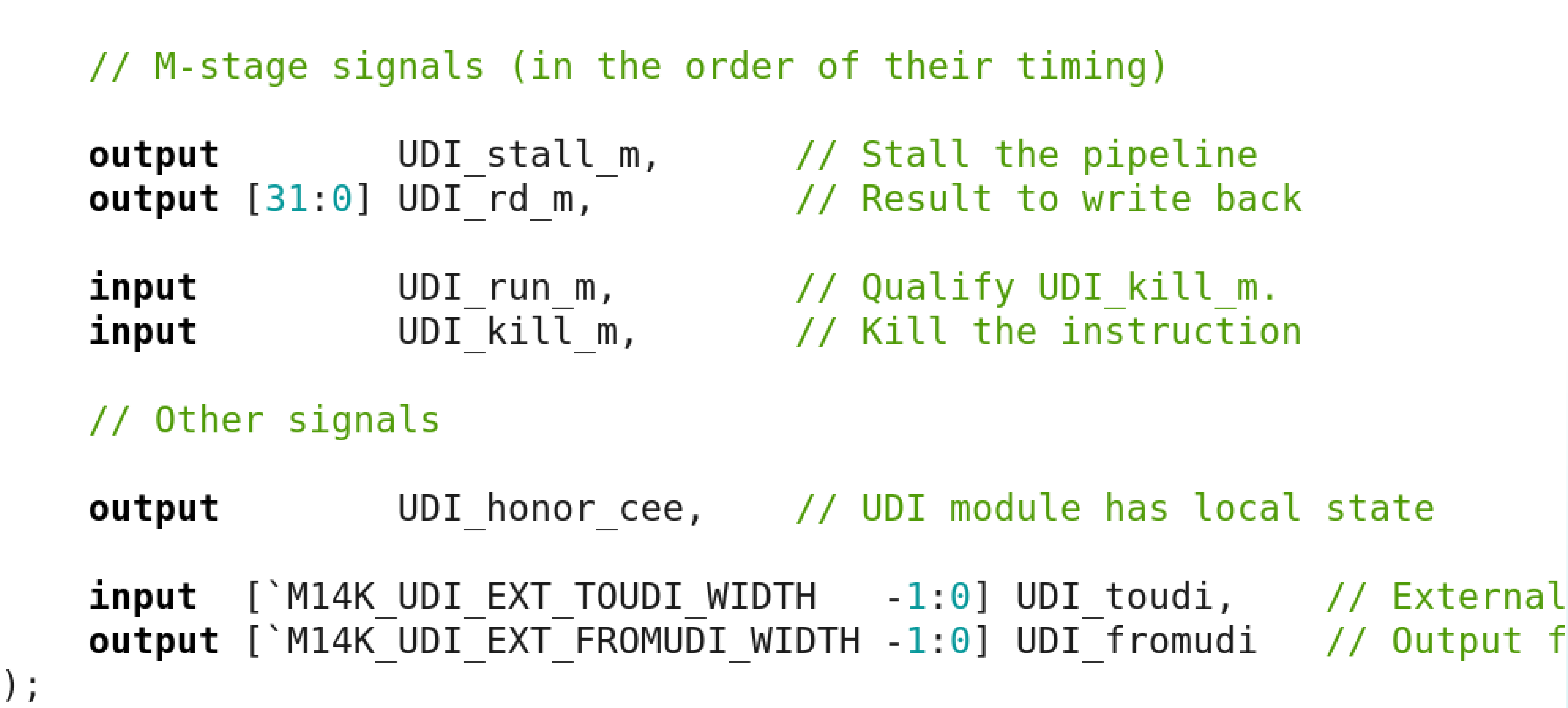
```
`include "m14k_const.vh"

module m14k_udi_mipsfpga_ai
(
    input         UDI_gclk,          // Clock
    input         UDI_greset,        // Reset
    input         UDI_gscanenable,   // Scan enable

    // Static signals

    input         UDI_endianb_e,     // Endian : 0 = little , 1 = big
    input         UDI_kd_mode_e,     // Mode   : 0 = user   , 1 = kernel or debug
    output        UDI_present,       // UDI module is present
```

CorExtend block code - 2



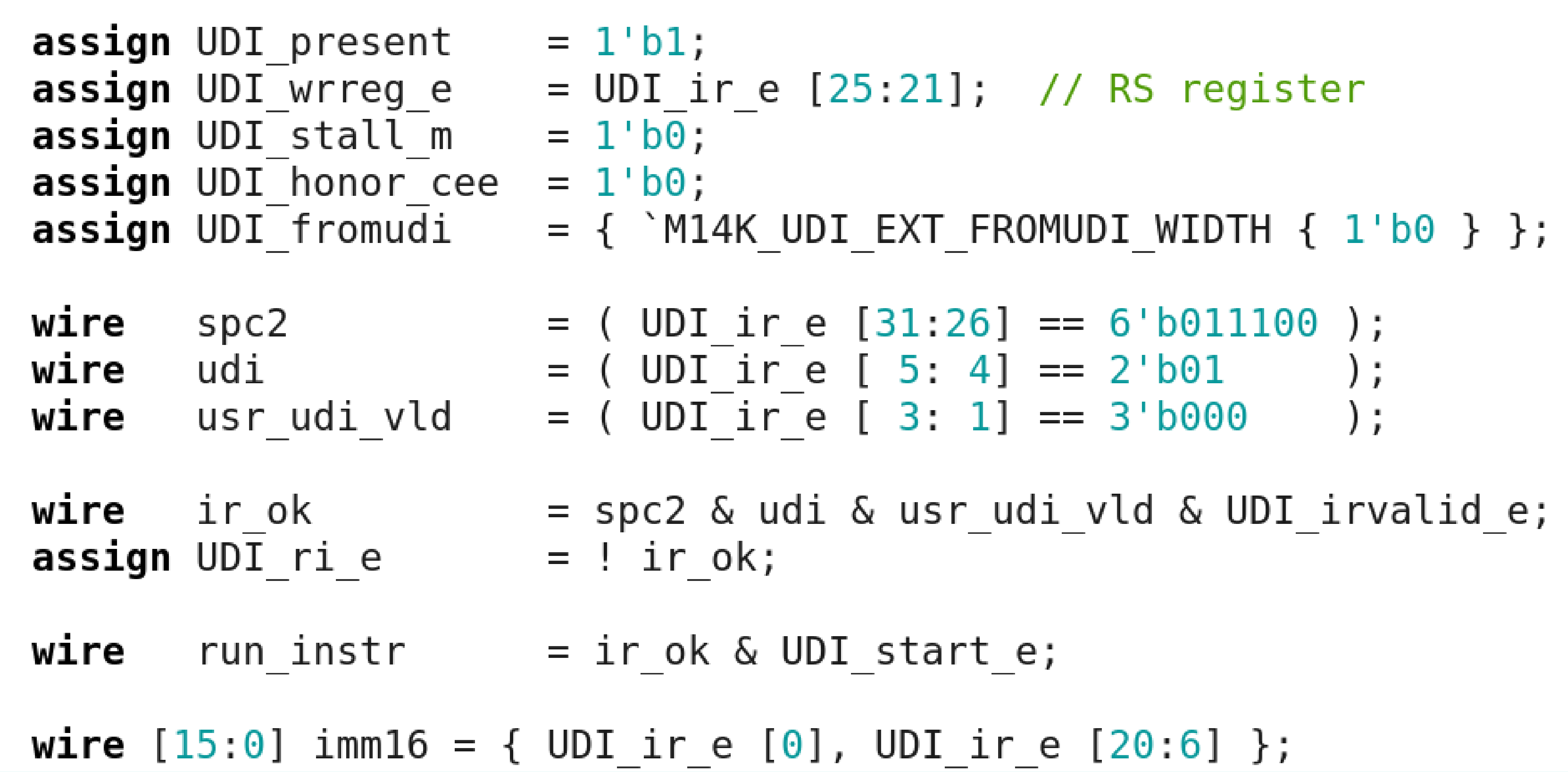
```
// E-Stage signals (in the order of their timing)

input  [31:0] UDI_ir_e,         // Instruction register
input         UDI_irvalid_e,    // Instruction register valid

output        UDI_ri_e,         // The instruction is illegal

input  [31:0] UDI_rs_e,         // Value of register RS from register file
input  [31:0] UDI_rt_e,         // Value of register RT from register file

output [ 4:0] UDI_wrreg_e,      // Register file index to write the result
                                // Zero index indicates don't write

input         UDI_start_e,      // Values of RS and RT valid
```

CorExtend block code - 3



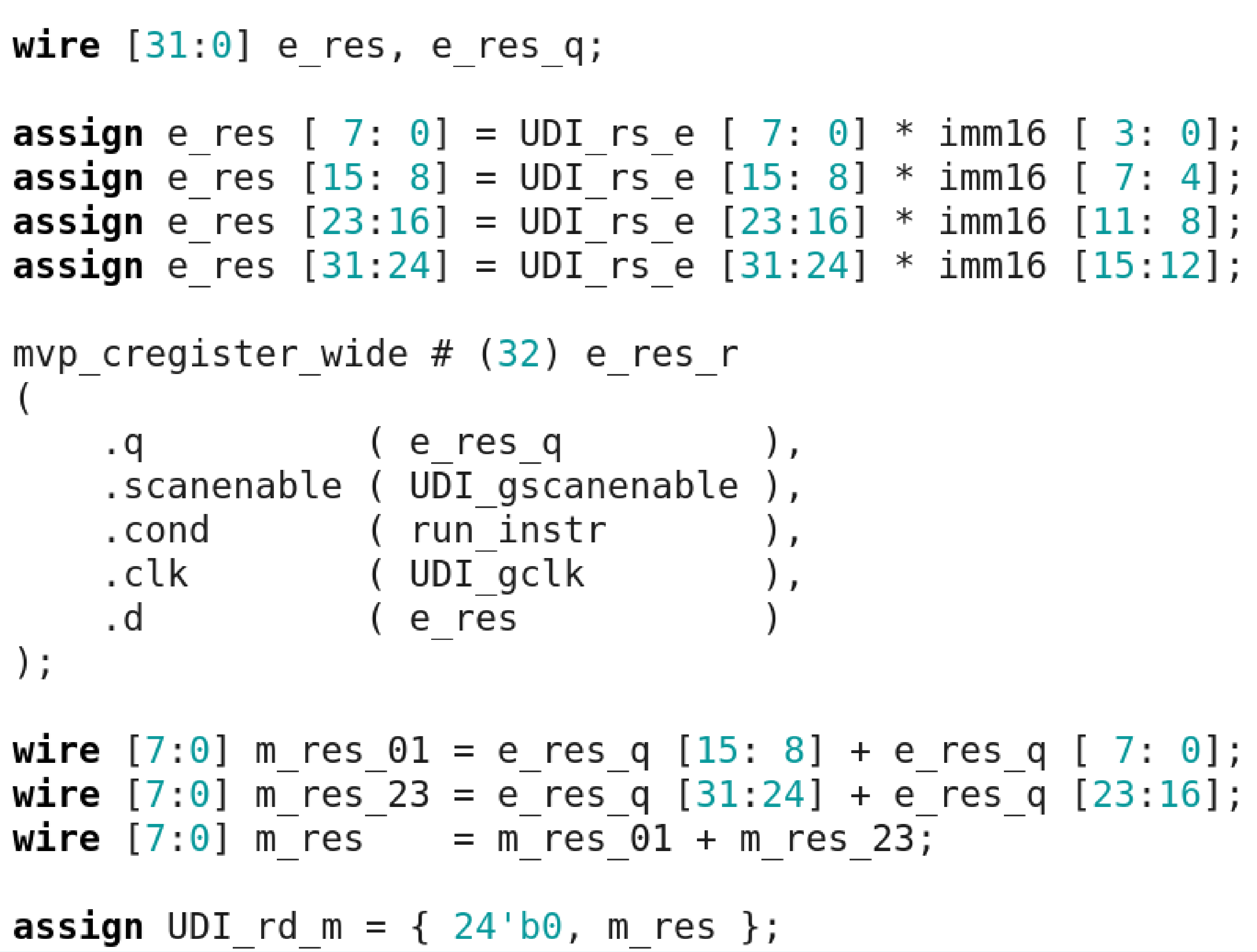
```
    // M-stage signals (in the order of their timing)

    output        UDI_stall_m,      // Stall the pipeline
    output [31:0] UDI_rd_m,         // Result to write back

    input         UDI_run_m,        // Qualify UDI_kill_m.
    input         UDI_kill_m,       // Kill the instruction

    // Other signals

    output        UDI_honor_cee,    // UDI module has local state

    input  [`M14K_UDI_EXT_TOUDI_WIDTH   -1:0] UDI_toudi,    // External
    output [`M14K_UDI_EXT_FROMUDI_WIDTH -1:0] UDI_fromudi   // Output f
);
```

CorExtend block code - 4



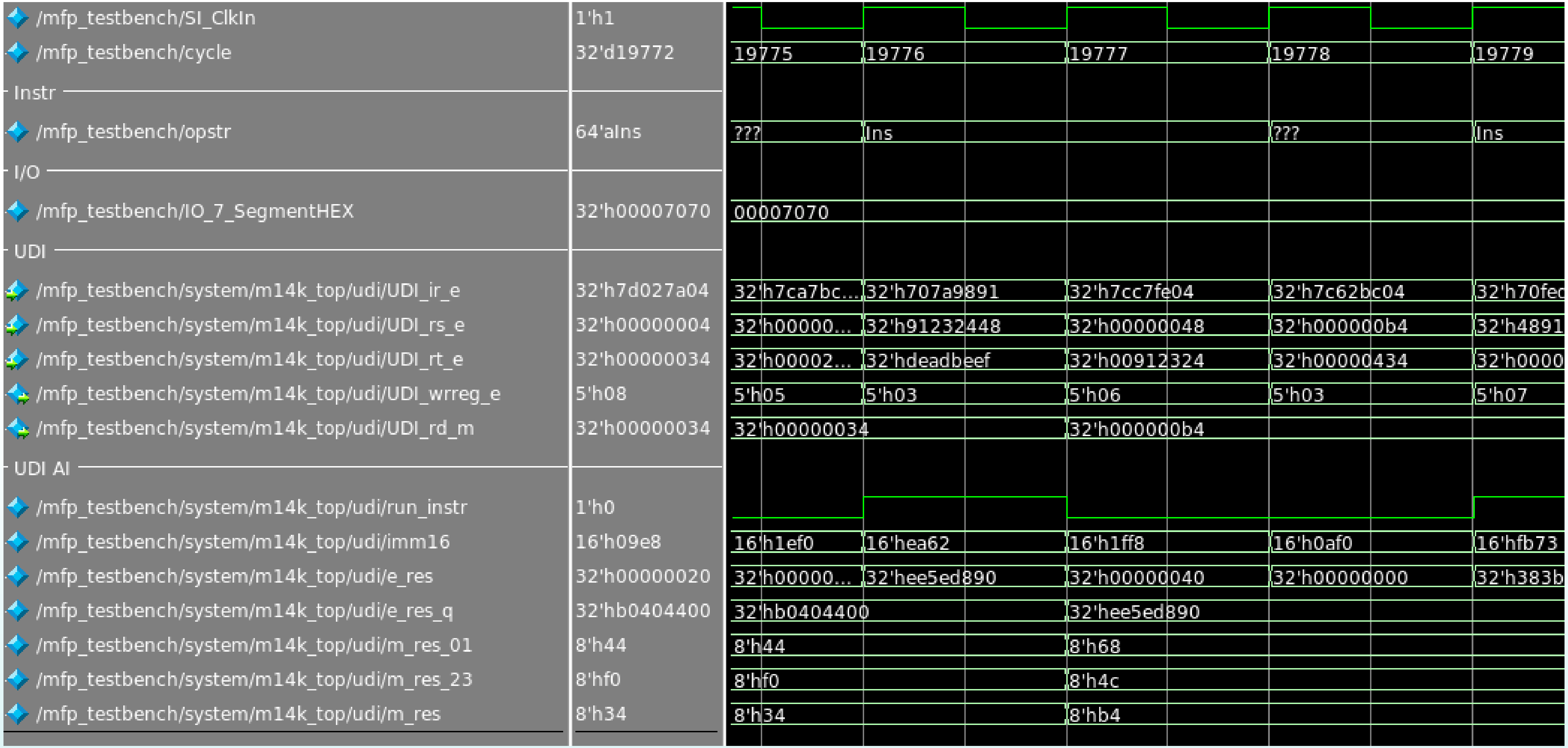
```
assign UDI_present    = 1'b1;
assign UDI_wrreg_e    = UDI_ir_e [25:21];  // RS register
assign UDI_stall_m    = 1'b0;
assign UDI_honor_cee  = 1'b0;
assign UDI_fromudi    = { `M14K_UDI_EXT_FROMUDI_WIDTH { 1'b0 } };

wire   spc2           = ( UDI_ir_e [31:26] == 6'b011100 );
wire   udi            = ( UDI_ir_e [ 5: 4] == 2'b01     );
wire   usr_udi_vld    = ( UDI_ir_e [ 3: 1] == 3'b000    );

wire   ir_ok          = spc2 & udi & usr_udi_vld & UDI_irvalid_e;
assign UDI_ri_e       = ! ir_ok;

wire   run_instr      = ir_ok & UDI_start_e;

wire [15:0] imm16 = { UDI_ir_e [0], UDI_ir_e [20:6] };
```

CorExtend block code - 5



```
wire [31:0] e_res, e_res_q;

assign e_res [ 7: 0] = UDI_rs_e [ 7: 0] * imm16 [ 3: 0];
assign e_res [15: 8] = UDI_rs_e [15: 8] * imm16 [ 7: 4];
assign e_res [23:16] = UDI_rs_e [23:16] * imm16 [11: 8];
assign e_res [31:24] = UDI_rs_e [31:24] * imm16 [15:12];

mvp_cregister_wide # (32) e_res_r
(
    .q           ( e_res_q          ),
    .scanenable  ( UDI_gscanenable ),
    .cond        ( run_instr        ),
    .clk         ( UDI_gclk         ),
    .d           ( e_res            )
);

wire [7:0] m_res_01 = e_res_q [15: 8] + e_res_q [ 7: 0];
wire [7:0] m_res_23 = e_res_q [31:24] + e_res_q [23:16];
wire [7:0] m_res    = m_res_01 + m_res_23;

assign UDI_rd_m = { 24'b0, m_res };
```

Simulate MIPS Open system using Verilog simulator

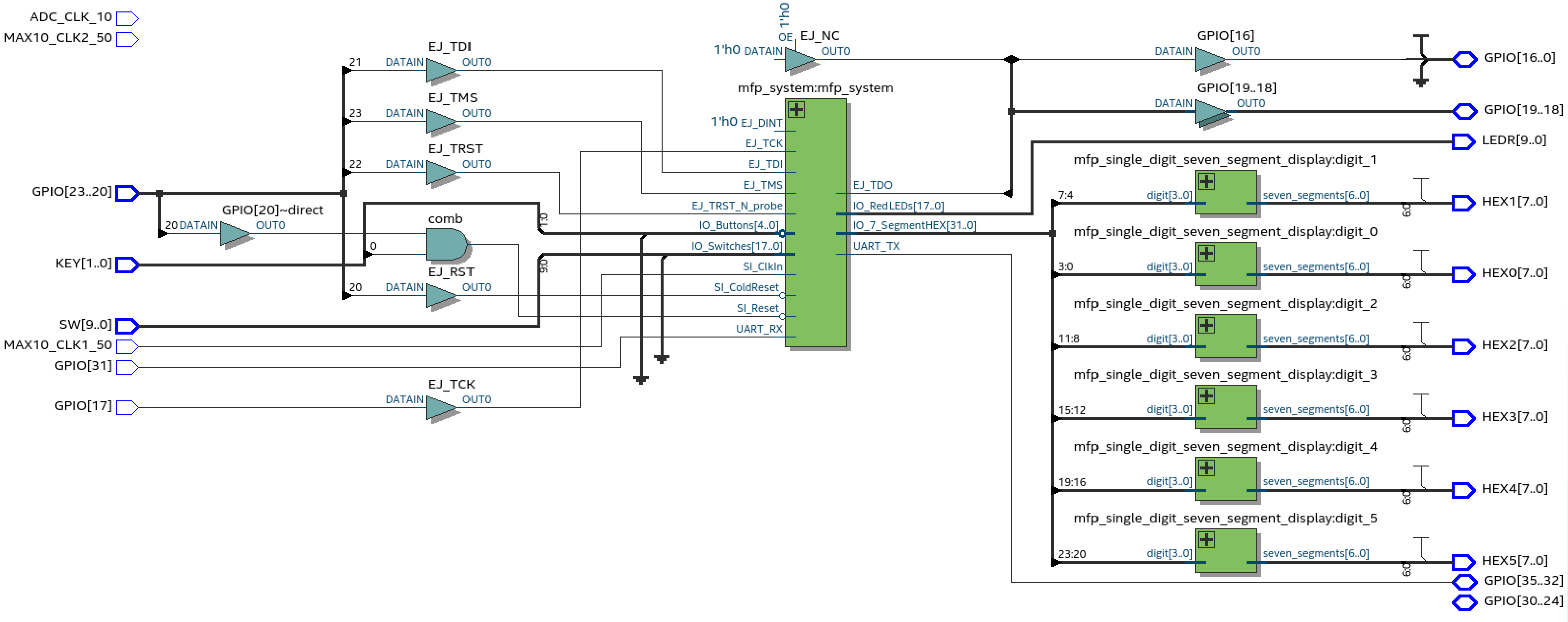


Signal	Value	19775	19776	19777	19778	19779
/mfp_testbench/SI_ClkIn	1'h1					
/mfp_testbench/cycle	32'd19772	19775	19776	19777	19778	19779
Instr						
/mfp_testbench/opstr	64'aIns	???	Ins		???	Ins
I/O						
/mfp_testbench/IO_7_SegmentHEX	32'h00007070	00007070				
UDI						
/mfp_testbench/system/m14k_top/udi/UDI_ir_e	32'h7d027a04	32'h7ca7bc...	32'h707a9891	32'h7cc7fe04	32'h7c62bc04	32'h70fed
/mfp_testbench/system/m14k_top/udi/UDI_rs_e	32'h00000004	32'h00000...	32'h91232448	32'h00000048	32'h000000b4	32'h4891
/mfp_testbench/system/m14k_top/udi/UDI_rt_e	32'h00000034	32'h00002...	32'hdeadbeef	32'h00912324	32'h00000434	32'h0000
/mfp_testbench/system/m14k_top/udi/UDI_wrreg_e	5'h08	5'h05	5'h03	5'h06	5'h03	5'h07
/mfp_testbench/system/m14k_top/udi/UDI_rd_m	32'h00000034	32'h00000034		32'h000000b4		
UDI AI						
/mfp_testbench/system/m14k_top/udi/run_instr	1'h0					
/mfp_testbench/system/m14k_top/udi/imm16	16'h09e8	16'h1ef0	16'hea62	16'h1ff8	16'h0af0	16'hfb73
/mfp_testbench/system/m14k_top/udi/e_res	32'h00000020	32'h00000...	32'hee5ed890	32'h00000040	32'h00000000	32'h383b
/mfp_testbench/system/m14k_top/udi/e_res_q	32'hb0404400	32'hb0404400		32'hee5ed890		
/mfp_testbench/system/m14k_top/udi/m_res_01	8'h44	8'h44		8'h68		
/mfp_testbench/system/m14k_top/udi/m_res_23	8'hf0	8'hf0		8'h4c		
/mfp_testbench/system/m14k_top/udi/m_res	8'h34	8'h34		8'hb4		

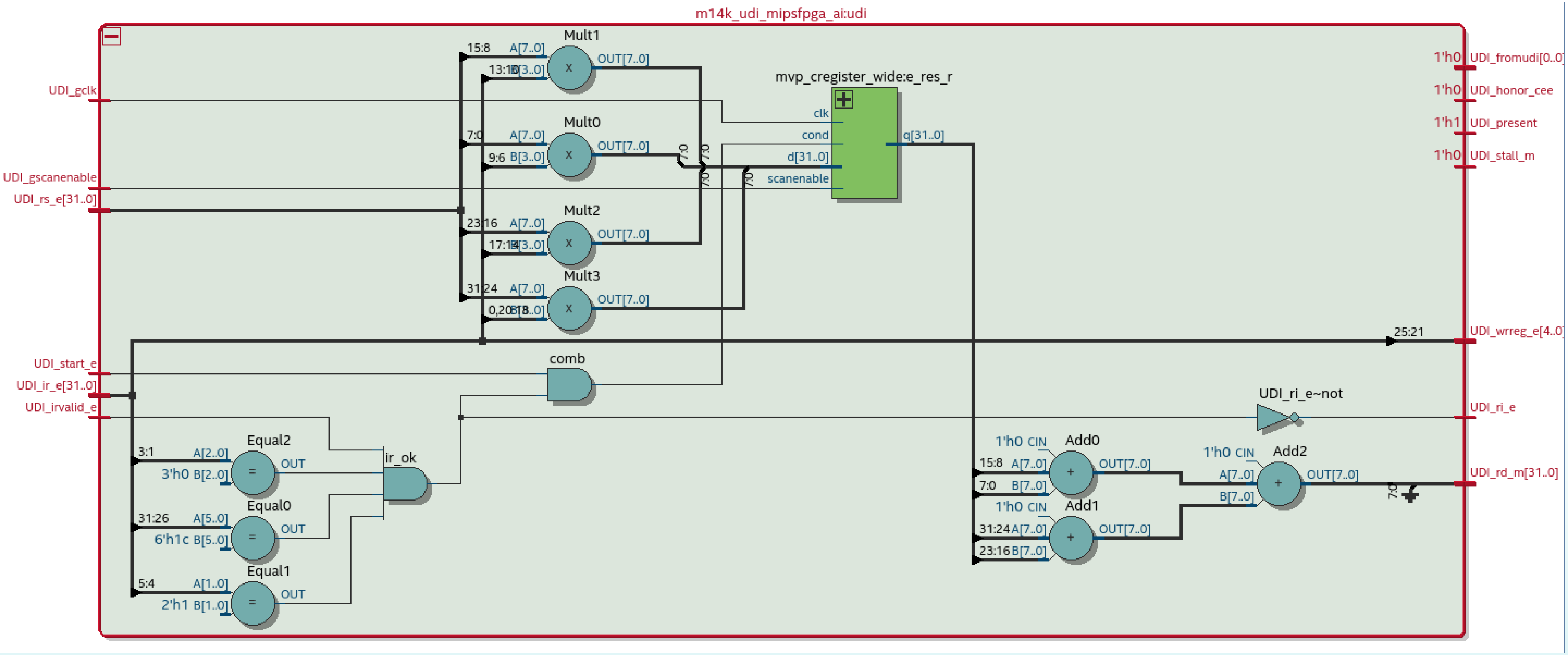
The results of the simulation

- The computation results matches
- Software computation takes 62 cycles
- Software-hardware computation takes 30 cycles - two times less
- The result can be improved orders of magnitude by making complicated AI engine that has both CorExtend and DSPRAM interfaces for highly parallel multi-functional computational unit

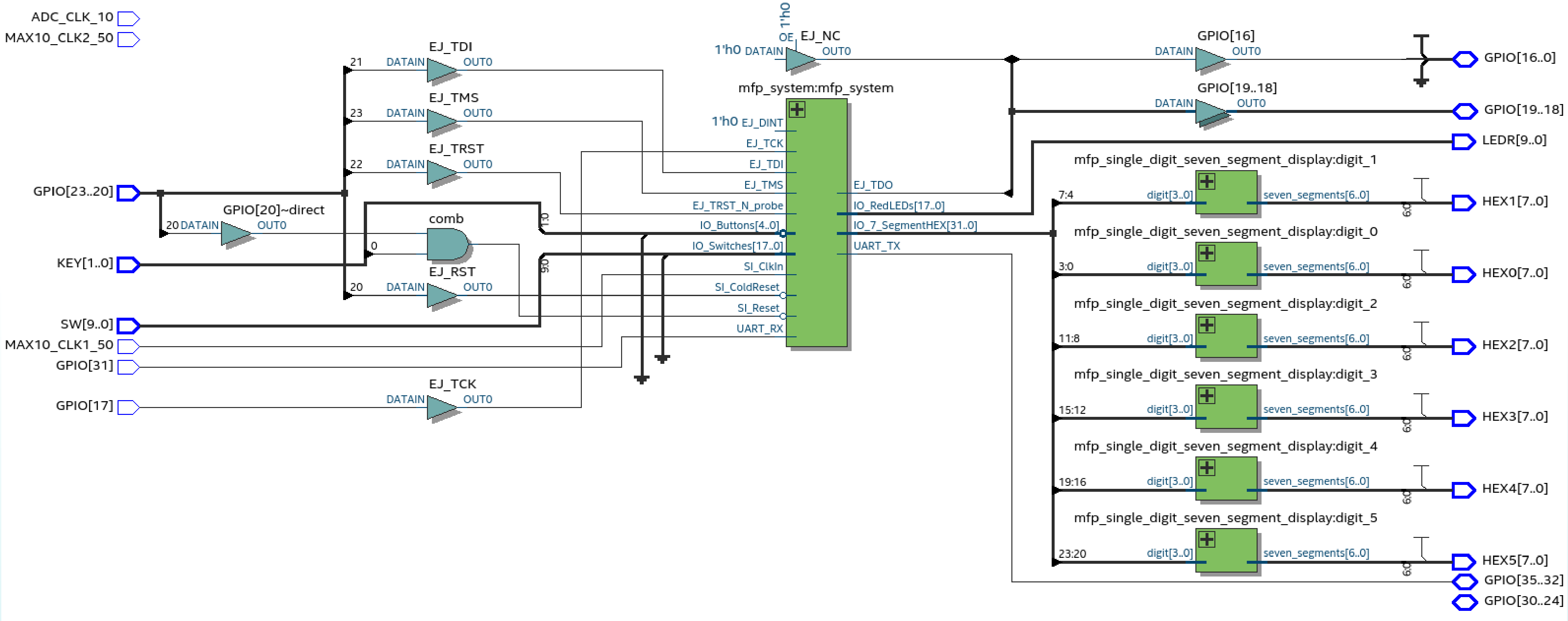
Synthesis – the whole MIPS Open FPGA system



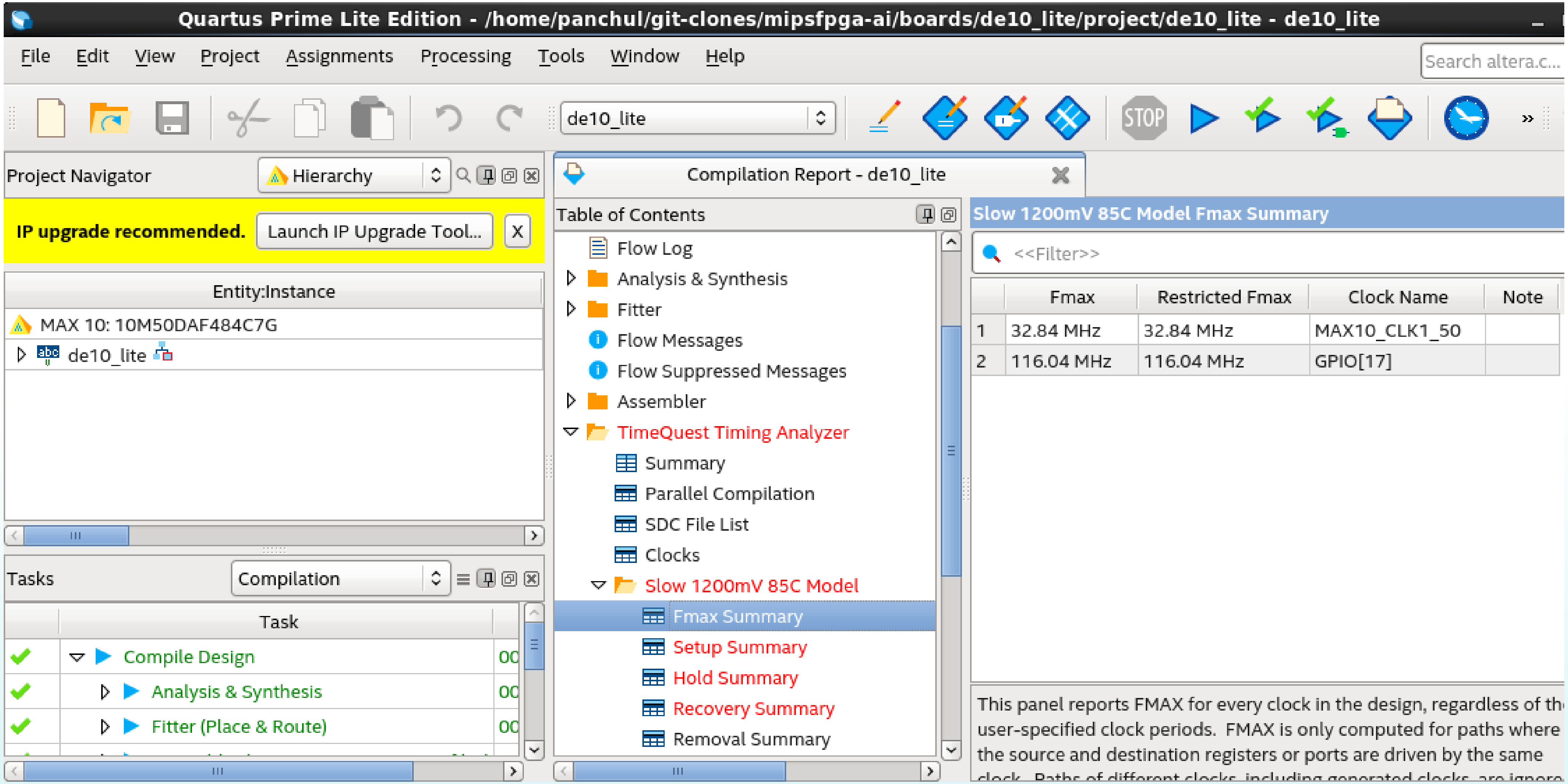
UDI submodule



Synthesis for the system
Fits 65% of Terasic DE10-Lite board



Fmax 33 MHz is practical even for Linux debug



	Fmax	Restricted Fmax	Clock Name	Note
1	32.84 MHz	32.84 MHz	MAX10_CLK1_50	
2	116.04 MHz	116.04 MHz	GPIO[17]	

This panel reports FMAX for every clock in the design, regardless of th user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignore

Housekeeping

- Return Hardware
  - Altera/Xilinx FPGA Boards
  - USB Hub
  - Cables
- Housekeeping
  - Please delete the Altera Quartus and Xilinx Vivado tools on the SSD!!!
  - When you leave the class, please leave the FPGA boards, cables and USB hub.
  - The SSD is yours to take home

WAVE® COMPUTING

Thank You

MIPS Open™