TramelBlaze Introduction

CECS 460

CSULB

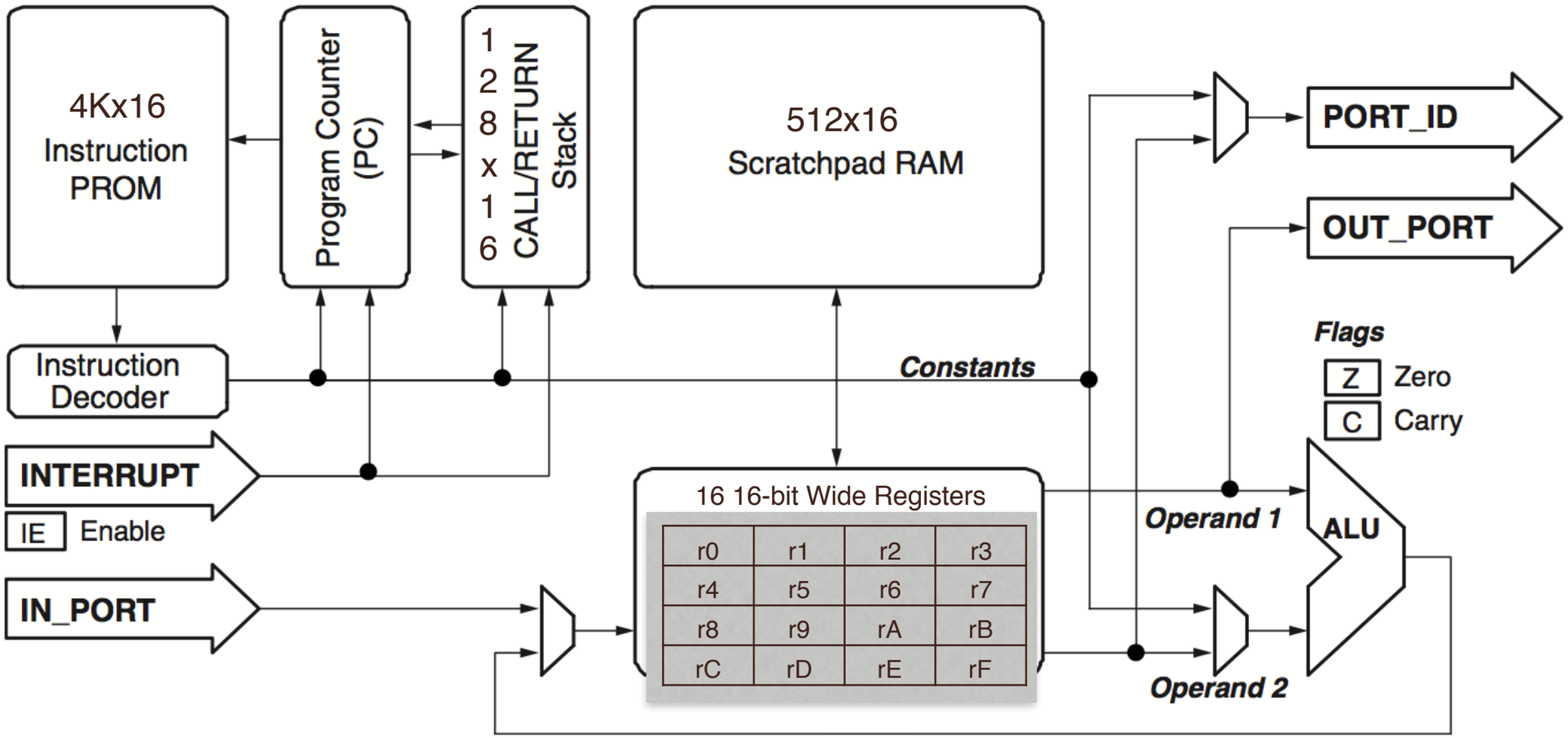
John Tramel

TramelBlaze

- The TramelBlaze is a 16-bit processor core designed to emulate the Xilinx PicoBlaze
- The instruction set is the PicoBlaze instruction set
- It comes as Verilog source
- It also comes with an Python Assembler for you to develop your code
- Your cooperation is requested in reporting any anomalies you might uncover during your use of the TramelBlaze

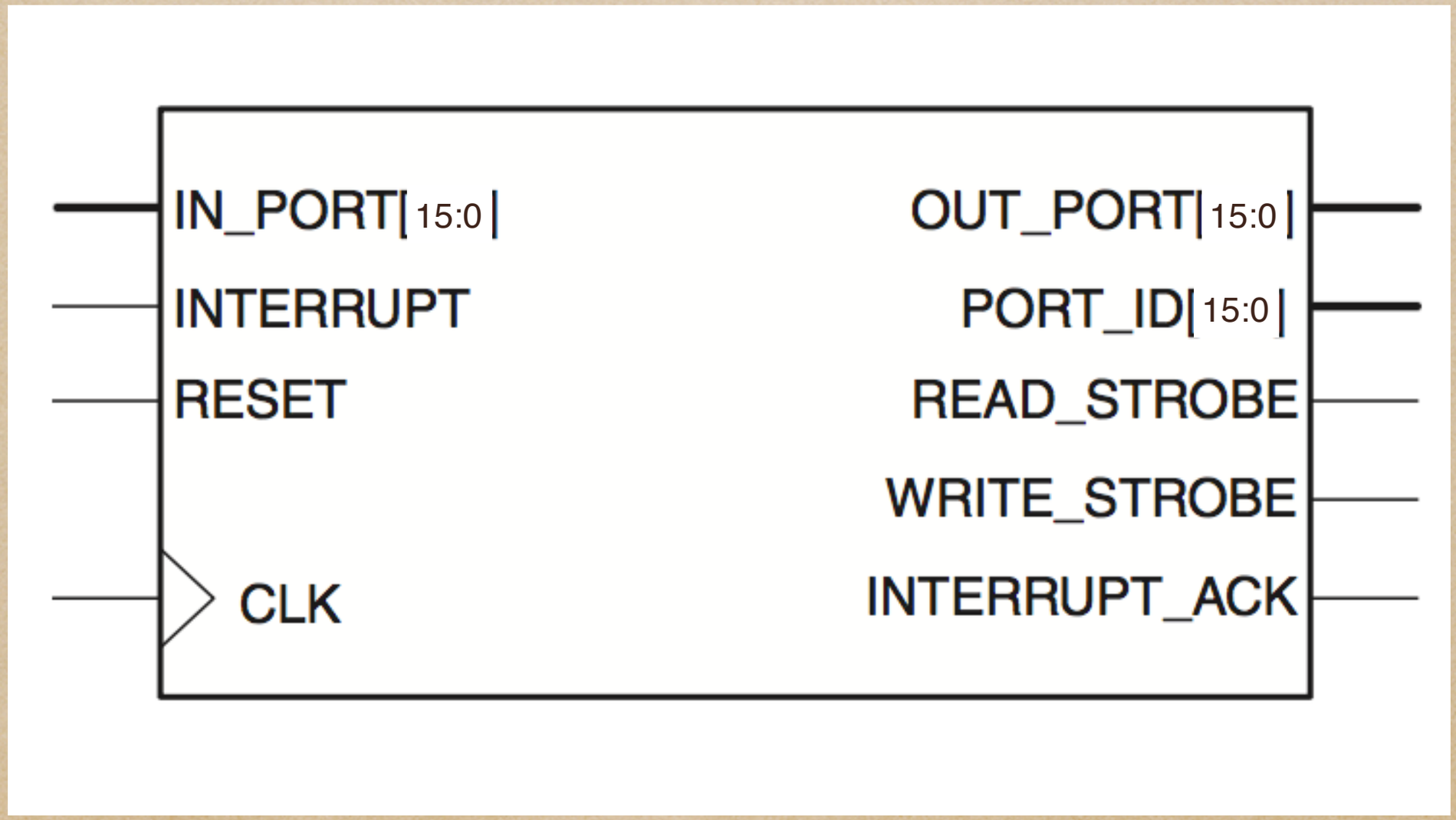
Tramelblaze Architecture

(16 bit emulator of 8 bit Picoblaze)



Top Level Block Diagram

Tramelblaze



Top Level TramelBlaze Instantiation

```
tramelblaze_top tbt   (.CLK(clk),
                       .RESET(rst),
                       .IN_PORT(inport),
                       .INTERRUPT(interrupt),

                       .OUT_PORT(outport),
                       .PORT_ID(portid),
                       .READ_STROBE(readstrobe),
                       .WRITE_STROBE(writestrobe),
                       .INTERRUPT_ACK(interruptack)
                       );
```

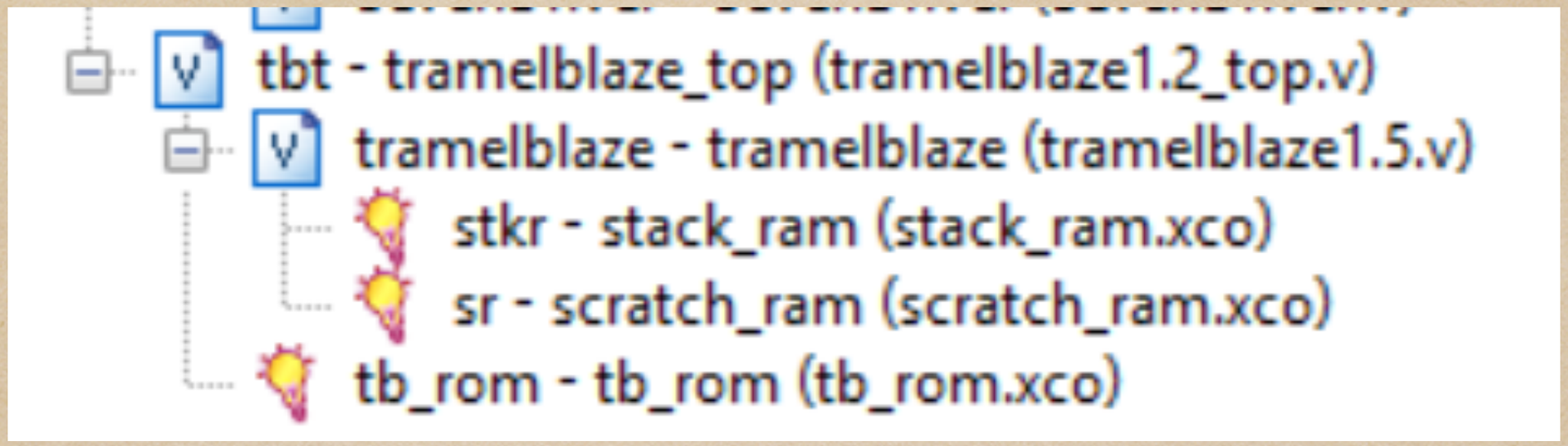
- This instantiation corresponds to the top level block diagram of the TramelBlaze
- This is what *you* instantiate in *your* design to include the TramelBlaze
- tramelblaze_top will instantiate the TramelBlaze processor and the instruction memory

TramelBlaze Instantiation within TramelBlaze_top

```
tramelblaze tramelblaze
        (
        .CLK(CLK),
        .RESET(RESET),
        .IN_PORT(IN_PORT),
        .INTERRUPT(INTERRUPT),

        .OUT_PORT(OUT_PORT),
        .PORT_ID(PORT_ID),

        .READ_STROBE(READ_STROBE),
        .WRITE_STROBE(WRITE_STROBE),
        .INTERRUPT_ACK(INTERRUPT_ACK),

        .ADDRESS(ADDRESS),
        .INSTRUCTION(INSTRUCTION)
        );

tb_rom tb_rom (
        .clka(CLK),                 // input clka
        .addra(ADDRESS),            // input  [11 : 0] addra
        .douta(INSTRUCTION)         // output [15 : 0] douta
        );
```

- The processor should be instantiated along with the memory file that is produced by the assembler
- Both are Verilog files

Required Files



- The hierarchy here is the tramelblaze_top module which instantiates the tramelblaze and tb_rom.
  - tramelblaze is the engine and tb_rom is the executable code saved in a "ROM"
- There are also three total memories that must be created: a stack RAM, a scratch RAM, and the aforementioned code ROM

Assembly Code 1

```
; background processing for tramelblaze
; successive addition and successive subtraction
; results are anticipated and compared in the code
; errors are displayed on the seven segment display
; meant to run while interrupts running to prove architecture

; python tramelblaze.py AddSub.tba

; 07 March 2016 john tramel cecs 460

; declare constants for coding

ZEROS           EQU     0000
ONE             EQU     0001
LOADLED         EQU     1233
COUNTLO         EQU     1234
COUNTHI         EQU     1235
ERRMUL          EQU     R8
ERRDIV          EQU     R9
COUNTMUL        EQU     R5
COUNTDIV        EQU     R6

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                           INITIALIZE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

START           ENINT

; initialize variables

                LOAD    R0,         ZEROS
                LOAD    ERRMUL,     ZEROS
                LOAD    ERRDIV,     ZEROS
                LOAD    COUNTMUL,   ZEROS
                LOAD    COUNTDIV,   ZEROS
```

- The assembly code is a text file where you write your code according to the PicoBlaze syntax
- Here is a sample piece of code declaring constants and initializing registers

Assembly Code 2

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                           MAIN LOOP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MAIN
; set up first MULSUB run

                LOAD    RD, 0068
                LOAD    RE, 005A
                LOAD    RF, 2490
                CALL    MULSUB
                ADD     COUNTMUL, ONE

; set up second MULSUB run

                LOAD    RD, 009B
                LOAD    RE, 00A3
                LOAD    RF, 62B1
                CALL    MULSUB
                ADD     COUNTMUL, ONE
                OUTPUT  COUNTMUL, COUNTHI

; set up first DIVSUB run

                LOAD    RD, 00AD
                LOAD    RE, 00AD
                LOAD    RF, 74E9
                CALL    DIVSUB
                ADD     COUNTDIV, ONE

; set up second DIVSUB run

                LOAD    RD, 0044
                LOAD    RE, 03C3
                LOAD    RF, FFCC
                CALL    DIVSUB
                ADD     COUNTDIV, ONE
                OUTPUT  COUNTDIV, COUNTLO

                JUMP    MAIN
```

- This continuation of the code defines the main loop of the program
- The main loop's operation will be interrupted whenever the tramelblaze services an interrupt

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                     INTERRUPT SERVICE ROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

                ADDRESS 0300

; set up for loop in isr

ISR
                ADD     R0, ONE
                OUTPUT  R0, LOADLED
                RETEN

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                           SUBROUTINES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; subroutine to perform multiplication by addition and check result
; RD: argument 1
; RE: argument 2
; RF: expected product

; RC: utilized in the routine for successive addition

MULSUB
                LOAD    RC, ZEROS
MULLOOP         ADD     RC, RD
                SUB     RE, ONE
                JUMPNZ  MULLOOP

; product now in RC - compare with expected value in RF

                COMP    RC, RF
                RETURNZ
                ADD     ERRMUL, ONE
                OUTPUT  ERRMUL, COUNTHI
                RETURN

; subroutine to perform division by subtraction and check result
; RD: argument 1
; RE: argument 2
; RF: expected product

; RC: utilized in the routine for successive subtraction

DIVSUB
                SUB     RF, RE
                SUB     RD, ONE
                JUMPNZ  DIVSUB

; we should have subtracted back to zero

                ADD     RF, RF          ; set zero flag
                RETURNZ
                ADD     ERRDIV, ONE
                OUTPUT  ERRDIV, COUNTLO
                RETURN
```

Assembly Code 3

- This continuation of the code contains the interrupt service routine and several subroutines

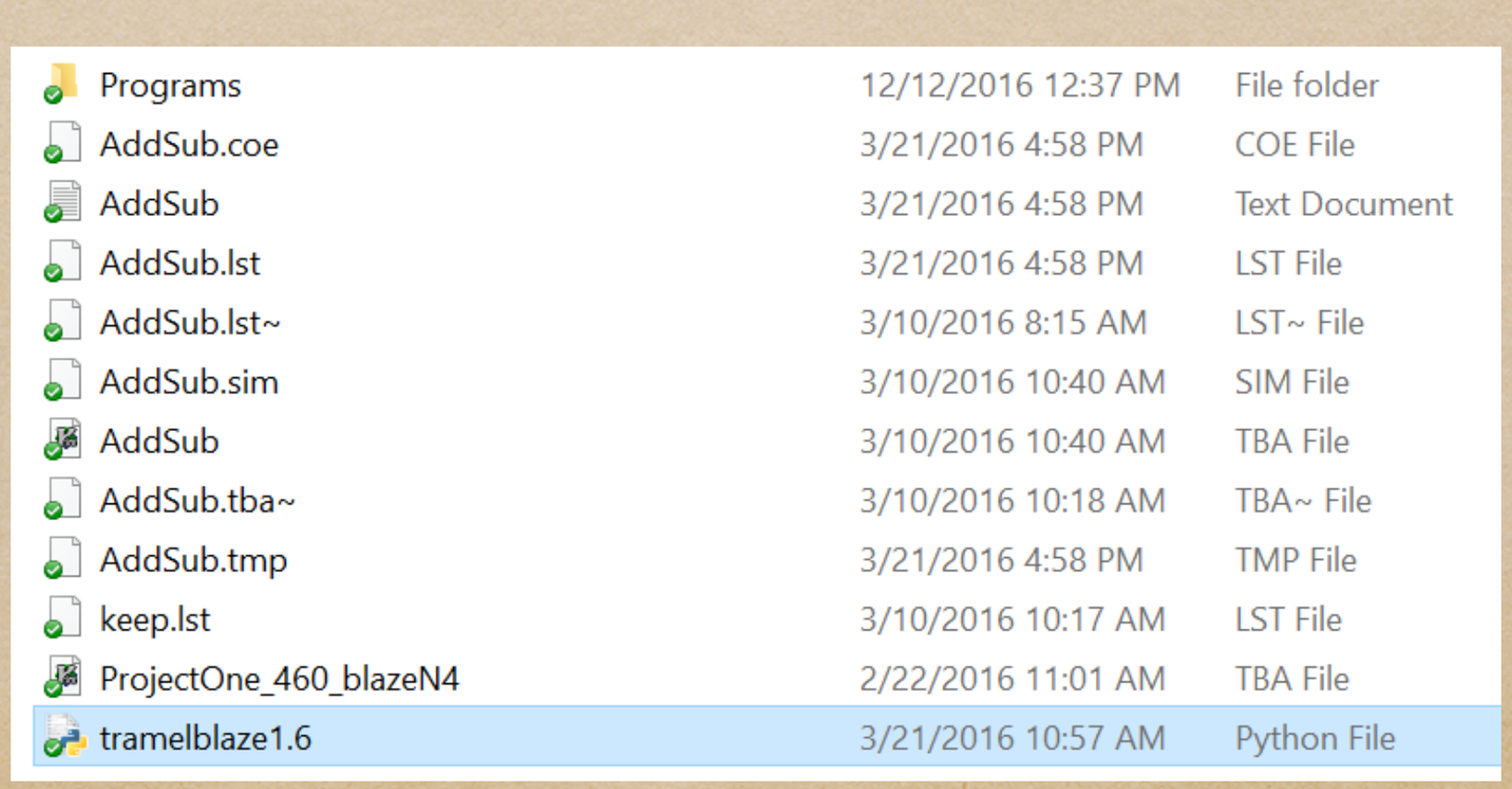
Assembly Code 4

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                         INTERRUPT VECTOR
;                         THROUGH 03FE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

              ADDRESS 0FFE

ENDIT         JUMP ISR

              END
```

- The last piece of code is found at the last two addresses: 3FE and 3FF. This is the address the interrupt will cause to be fetched and executed upon an interrupt occurrence

Assembler

- The assembler with the package is a Python program that you may run once you have installed Python. Download Python, if needed, from python.org
- The source file should be a text file with the .tba extension (Tramel Blaze Assembly)
- The assembler creates the .lst to assist in debug and the .coe for inclusion into the design



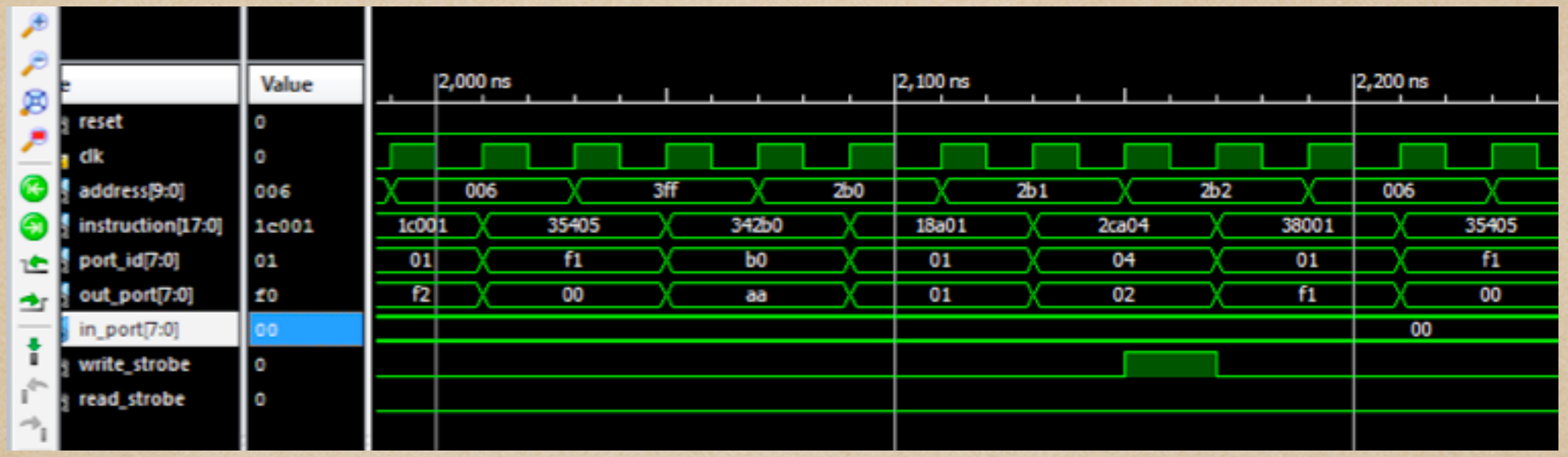
Programs	12/12/2016 12:37 PM	File folder
AddSub.coe	3/21/2016 4:58 PM	COE File
AddSub	3/21/2016 4:58 PM	Text Document
AddSub.lst	3/21/2016 4:58 PM	LST File
AddSub.lst~	3/10/2016 8:15 AM	LST~ File
AddSub.sim	3/10/2016 10:40 AM	SIM File
AddSub	3/10/2016 10:40 AM	TBA File
AddSub.tba~	3/10/2016 10:18 AM	TBA~ File
AddSub.tmp	3/21/2016 4:58 PM	TMP File
keep.lst	3/10/2016 10:17 AM	LST File
ProjectOne_460_blazeN4	2/22/2016 11:01 AM	TBA File
tramelblaze1.6	3/21/2016 10:57 AM	Python File

Machine Code

```
memory_initialization_radix = 16;
memory_initialization_vector =
1E00 AE00 0000 AE08 0000 AE09 0000 AE05 0000 AE06 0000 AE0D 0068 AE0E 005A AE0F
2490 8E00 0305 8205 0001 AE0D 009B AE0E 00A3 AE0F 62B1 8E00 0305 8205 0001 B805
1235 AE0D 00AD AE0E 00AD AE0F 74E9 8E00 0313 8206 0001 AE0D 0044 AE0E 03C3 AE0F
FFCC 8E00 0313 8206 0001 B806 1234 A400 000b 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

- The file that is created for the machine code is the .coe file. This code is in the format recognized by the Xilinx ISE when creating ROM memories.
- Every time you modify your source code and assemble it you will need to include the new file into the build
- Please refer to the Youtube videos for instructions

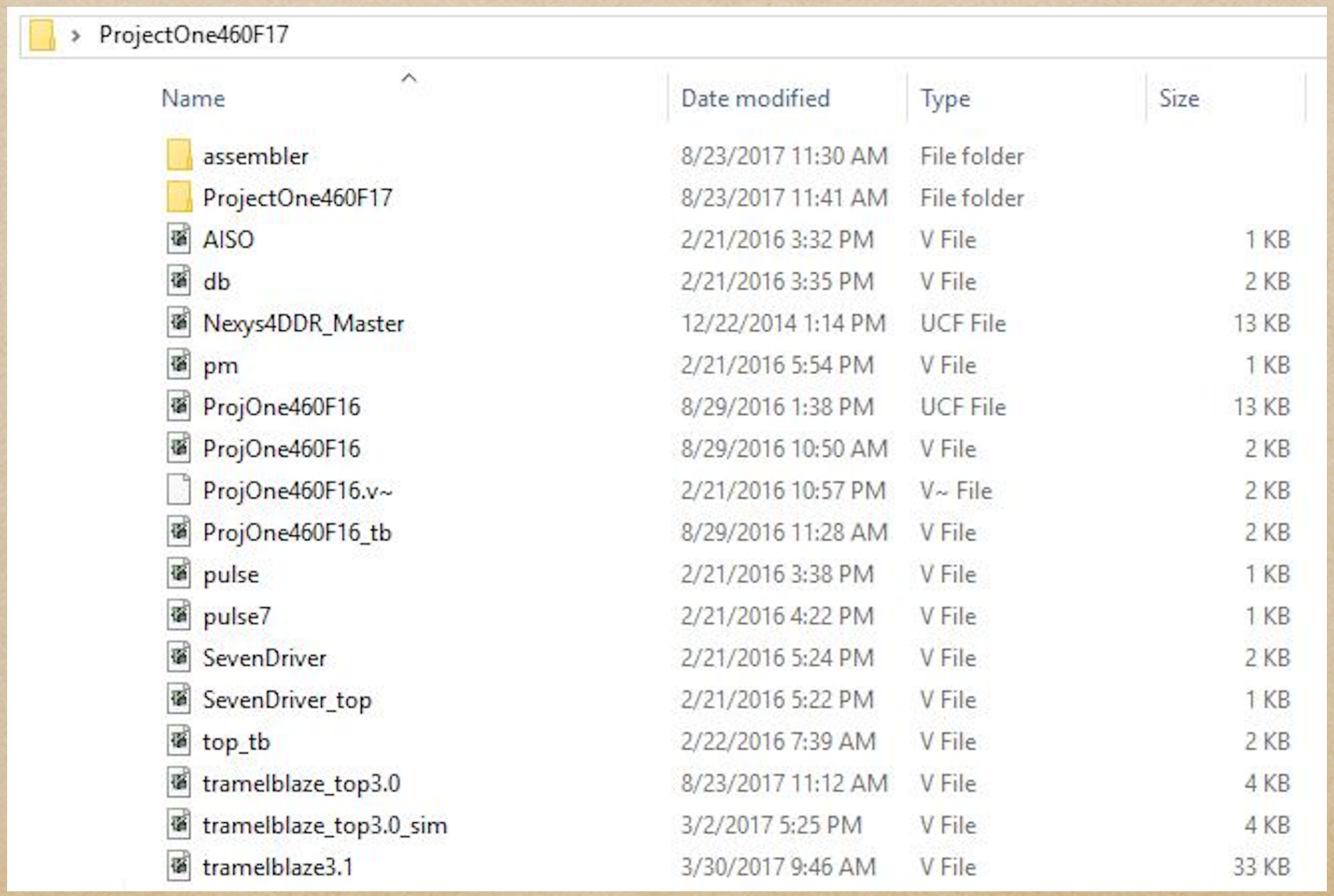
Simulation



- When you run the simulation you should see the PicoBlaze fetching and executing instructions and you should see the outputs of the PicoBlaze switching at the proper times
- A simple case like this is a quick way to ensure that your environment is set up properly

Development Hierarchy

Working Project (directory)
  assembler (directory)
  Working Project (directory)
  Project files

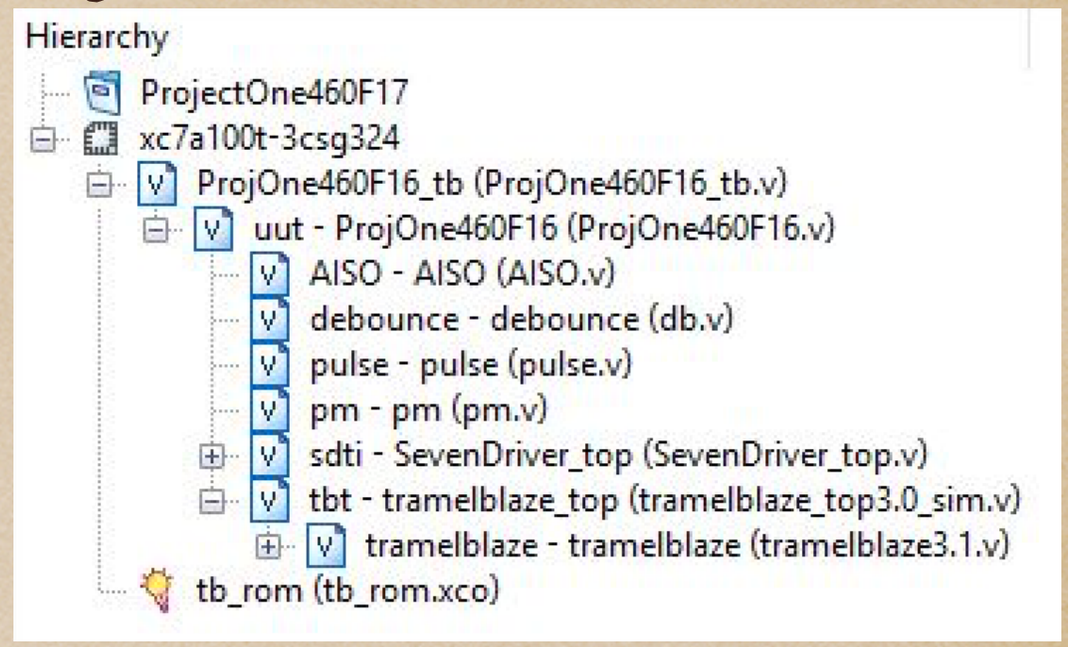


Development Flow

- There are two approaches for utilizing the tramelblaze in a design. The first is during development when the focus is on simulation and the second is verification on the hardware where the focus is running on the board
- In both approaches simulations may be run
- Whenever a design is destined to run on the board the ROM, tb_rom, must be created. This takes a little time that might be considered inconvenient during development. To facilitate development a second tramelblaze_top has been created called “tramelblaze_top_sim”
- This allows simulations to be run immediately without having to build the file tb_rom repeatedly
- You must utilize the hierarchy presented in this presentation in order for this scheme to work

New Design Hierarchy for Simulation

- Notice that tb_rom is no longer a part of the design. We will use a memory inside the tramelblaze top for simulation



- The simulation version defines a memory and will copy the sim.sim file produced by the assembler into the memory.
- Just re-assemble the code and run the simulator from time 0 and the new code file will be loaded

```
tramelblaze tramelblaze
      (
      .CLK(CLK),
      .RESET(RESET),
      .IN_PORT(IN_PORT),
      .INTERRUPT(INTERRUPT),

      .OUT_PORT(OUT_PORT),
      .PORT_ID(PORT_ID),
      .READ_STROBE(READ_STROBE),
      .WRITE_STROBE(WRITE_STROBE),
      .INTERRUPT_ACK(INTERRUPT_ACK),

      .ADDRESS(ADDRESS),
      .INSTRUCTION(INSTRUCTION)
      );

   reg [15:0] memory [0:4095];

   initial
      $readmemh("../Assembler/sim.sim",memory);

   always @(posedge CLK)
      INSTRUCTION = memory[ADDRESS];
```

YouTube Videos

- There are several YouTube videos available to help the transition to using the tramelblaze and writing assembly code
- You are expected to watch the videos and understand their content

```
tramelblaze start synced
   - introduction to tramelblaze

PongChu_ch16
PongChu_ch17
PongChu_ch18
   - Tramel reads Pong Chu's chapters on the
     processor and the assembler
```