Modeling the MSP430 in Verilog

Week 3

Victoria Rodriguez

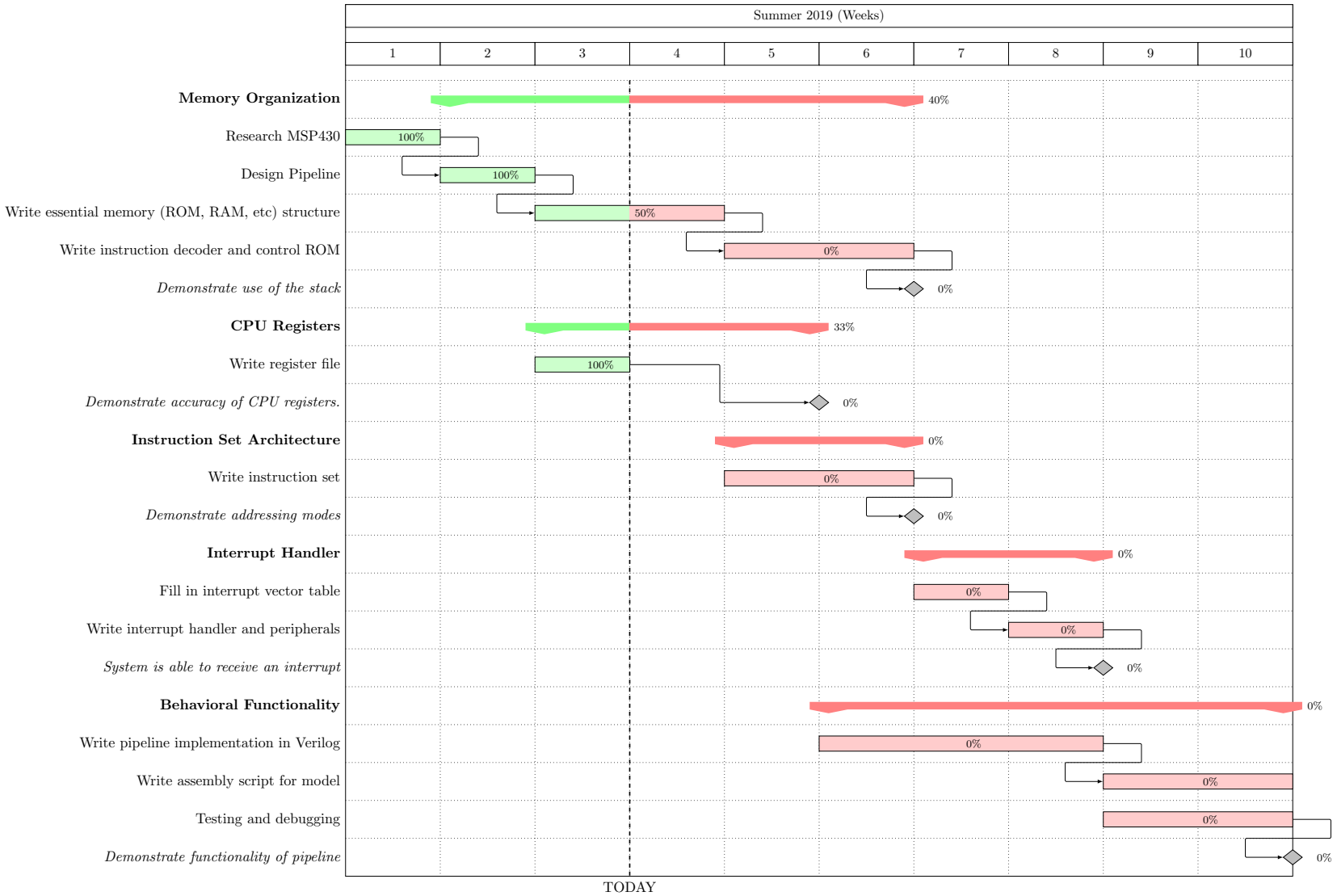
Texas Tech University

Wednesday 26th June, 2019

Overview

Logistics

Gantt Chart



Last week's deliverable was to write the register file.

Designing the Register File

Register File Objectives

R15	
R14	
R13	
R12	
R11	
R10	
R9	
R8	
R7	
R6	
R5	
R4	
R3 (CR2)	Constant Generator
R2 (SR/CR1)	Status Register
R1 (SP)	Stack Pointer
R0 (PC)	Program Counter

- All registers are addressable
- CPU registers line PC, SP, SR live in register file
- PC and SP always point to valid addresses in memory
- Constant generators work

System CPU Reset

[1]

- PC checks the reset vector for next address [2]
- Reset vector is content stored in FFFEh in Interrupt Vector Table (IVT)
- This means a memory access occurs every reset
- CPU reset caused by attempt to IF from 0000h–01FFh [3]

Algorithm

Algorithm 1 Register File

Input: Sys. Clock (`clk`), Reset (`rst`), Reset Vector (`RST_VEC`), Register Write (`RW`), Addressing Mode (`As`), PC in, SR in, SP in, Data In (`Din`), Source Address (`SA`), Destination Address (`DA`)

Output: PC out, SR out, SP out, Data Out (`Dout`), Source Out (`Sout`)

```
Create 2D 16x16 vector bus for the registers.
PC out ← R[0], SP out ← R[1], SR out ← R[2]
Sout ← R[SA], Dout ← R[DA]
for all positive edges of clk do
    if rst then
        R[2]–R[15] ← 0
        R[0] ← RST_VEC
    end if
    if RW then
        R[DA] ← Din
    end if
    if write to PC AND Din is valid then
        R[0] ← Din
    else if write to PC AND Din is NOT valid then
        R[0] ← RST_VEC
    else
        R[0] ← PC in
    end if
end for
```

Algorithm (Cont...)

Algorithm 1 Register File (Cont...)

```
if As is 2b00 AND SRC is R2 then
    R[SA] ← SR in
else if As is 2b01 AND SRC is R2 then
    R[SA] ← 0
else if As is 2b10 AND SRC is R2 then
    R[SA] ← 0h0004
else if As is 2b11 AND SRC is R2 then
    R[SA] ← 0h0008
else if As is 2b00 AND SRC is R3 then
    R[SA] ← 0h0001
else if As is 2b01 AND SRC is R3 then
    R[SA] ← 0h0002
else if As is 2b10 AND SRC is R3 then
    R[SA] ← 0hFFFF
else
    R2 ← SR in
end if
```

Implementing the Register File

Initializing Registers

```
// Initialize registers
reg [15:0]      regs [15:0];
integer         i;

initial
  begin
      for (i=0;i<16;i=i+1)
         regs[i] = 0;
  end
```

[4]

- Syntax is that first index is width, second is depth [5]
- Memory cannot be initialized all at once, so a `for`-loop is used to clear out all registers

Handling Outputs

```
assign reg_PC_out        = regs[0];
assign reg_SP_out        = regs[1];
assign reg_SR_out        = regs[2];
// Addressable registers
assign {Sout,Dout} = {regs[SA],regs[DA]};
```

[4]

Synchronous Write

```
always @ (posedge clk)
  begin
    if (rst)
      begin
        for (i=0;i<16;i=i+1)
          regs[i] <= 0;
        regs[0] <= RST_VEC; // ROM[FFFE]
      end
    // Write to registers
    else if (RW)
      regs[DA] <= Din;
  end // always @ (posedge clk)
```

[4]

- Reset clears all registers out, forces the reset vector into PC
- Reset vector comes from MDB because of memory access
- Writes to reg on clock

Latches

CPU Registers

Outgoing CPU registers come from latching the incoming CPU register. Output is one clock tick later.

```
// Conditional bits
wire        valid_Din_PC = (Din > 16'h01FF)     ? 1 : 0;
wire        write_to_PC = (!DA && RW)           ? 1 : 0;
wire        write_to_SP = ((DA == 4'd1) && RW)  ? 1 : 0;

// Increment PC happens inside of MUX PC
always @ (posedge clk)
  begin
    // Latch the incoming PC and SP
    regs[0] <= (write_to_PC && valid_Din_PC)  ? Din     :
               (write_to_PC && ~valid_Din_PC) ? RST_VEC : reg_PC_in;
    regs[1] <= (write_to_SP)                  ? Din     : reg_SP_in;
  end
```

[4]

SR Special Cases

Constant Generators

Addressing into R2 returns a constant. Otherwise it holds SR. R3 is always a constant.

```
// SR special cases
always @ (*)
  case({As,SA})
    // CONSTANTS GENERATED FROM R2
    {2'b00,4'd2}: regs[SA] <= reg_SR_in;
    {2'b01,4'd2}: regs[SA] <= 0;
    {2'b10,4'd2}: regs[SA] <= 16'h0004;
    {2'b11,4'd2}: regs[SA] <= 16'h0008;
    // CONSTANTS GENERATED FROM R3
    {2'b00,4'd3}: regs[SA] <= 0;
    {2'b01,4'd3}: regs[SA] <= 16'h0001;
    {2'b10,4'd3}: regs[SA] <= 16'h0002;
    {2'b11,4'd3}: regs[SA] <= 16'hFFFF;
    default: regs[2] <= reg_SR_in;
  endcase
```

[4]

Simulation Results

Testbench: Initialization

```
initial
  begin
      clk <= 0;
     RW  <= 0;
      rst <= 0;
      {DA, SA, As} <= 0;
     RST_VEC <= 16'hC000;
      reg_PC_in <= 16'hC000;
      reg_SP_in <= 16'h0200;
      reg_SR_in <= 16'hFFFF;
      Din <= 16'hF000;
      #75 Din <= 16'h0;
  end
```

[4]

- Start address of ROM is C000h [3]
- Start address of RAM is 0200h [3]

Testbench: Emulated Signals

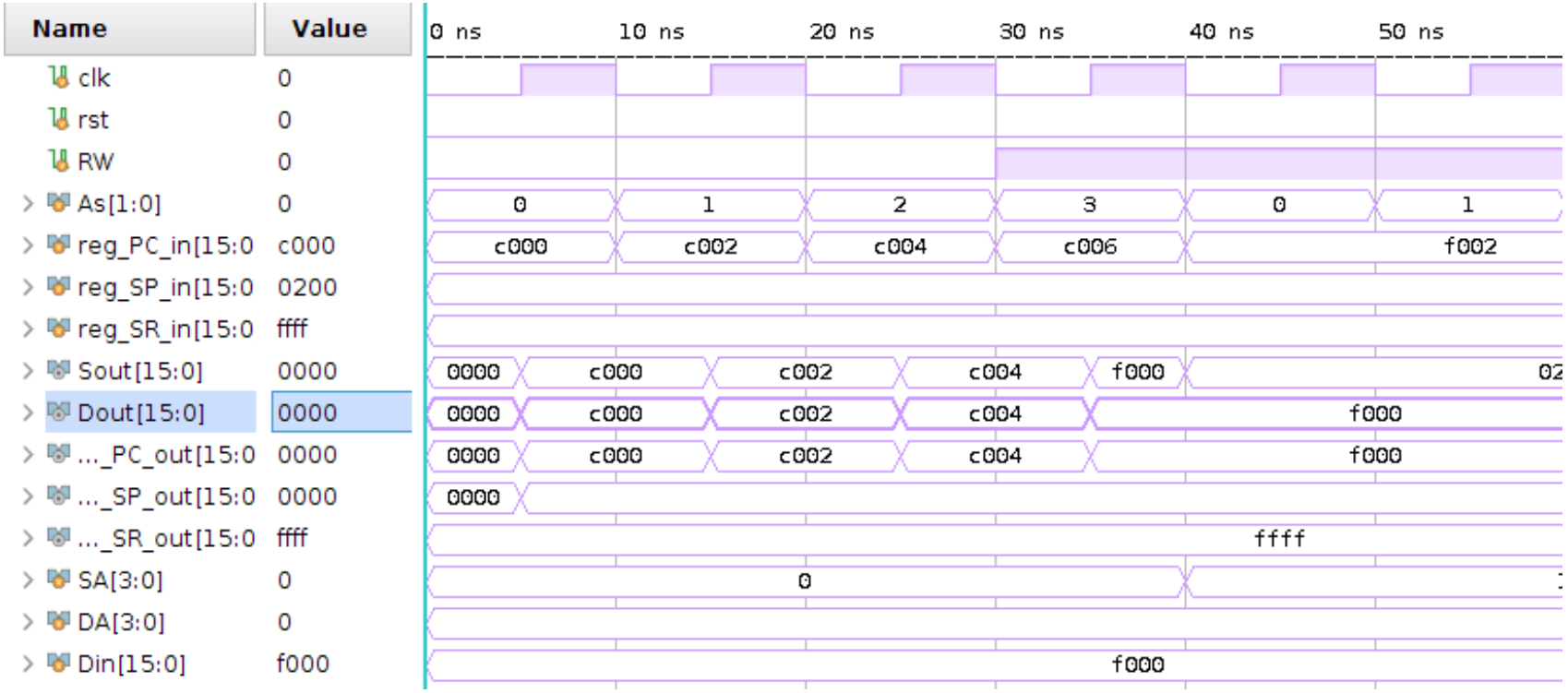
PC is Forced Even

PC is forced to an even address by asserting LSB as 0. Later this will be implemented in the MUX PC.

```
always #5 clk = ~clk;
always #10 {DA, SA, As} = {DA, SA, As} + 1;
always #10 reg_PC_in = {reg_PC_out[15:1],1'b0} + 2;
always #30 RW <= ~RW;
```

[4]

Waveform: PC Write



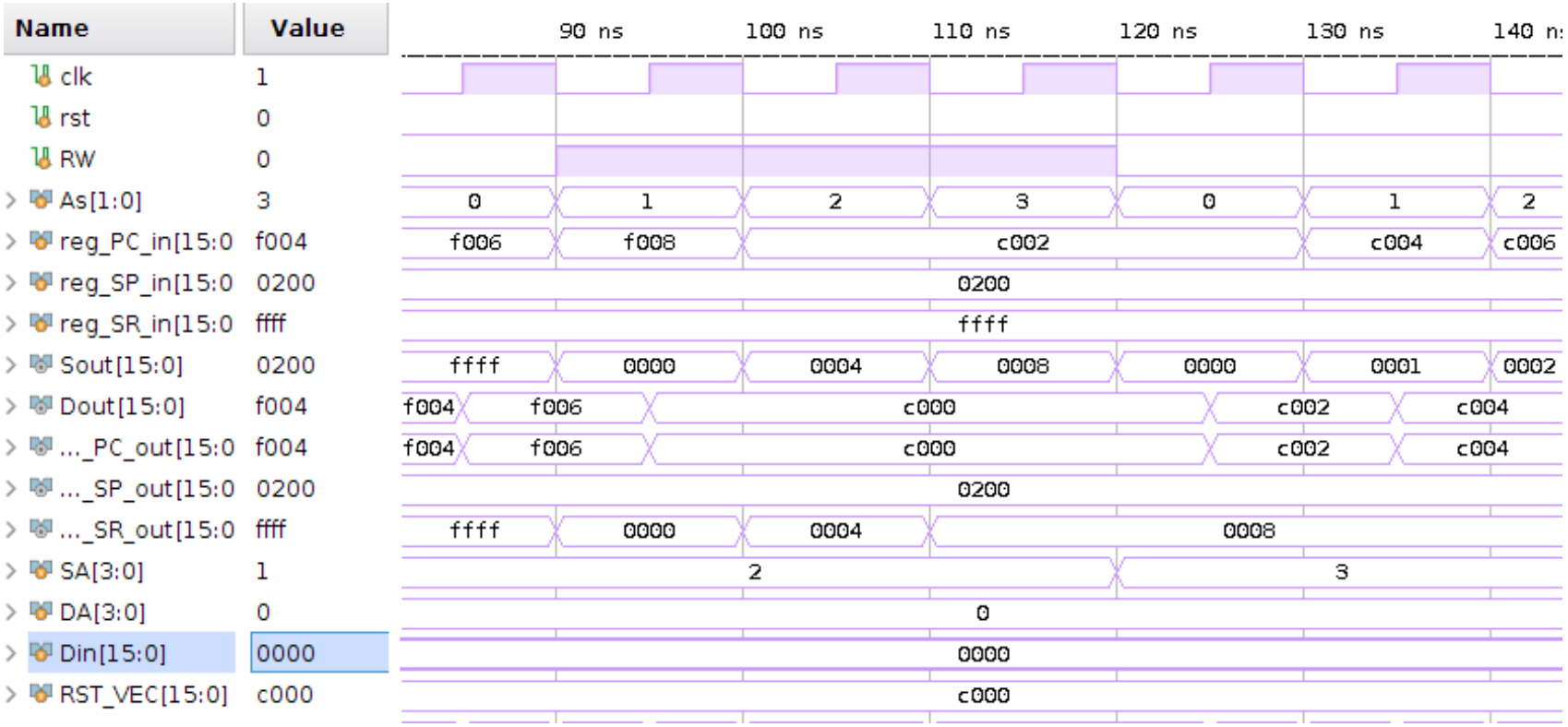
Writes to PC

The user is able to write to PC if it is a valid address.

Waveform: Reset if Invalid

Sys. Reset

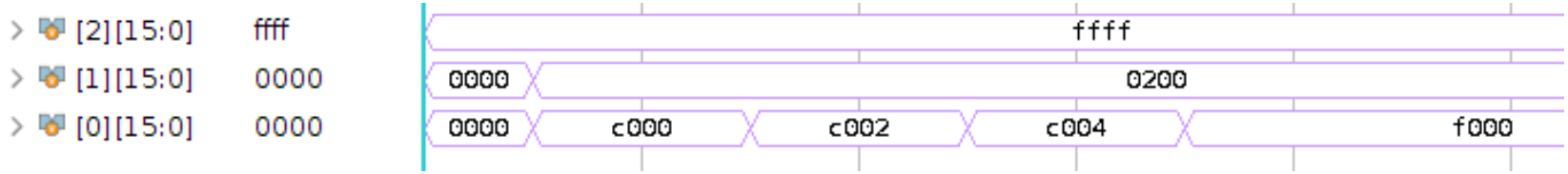
Latches to reset vector if `Din` is not a valid address



Waveform: Registers

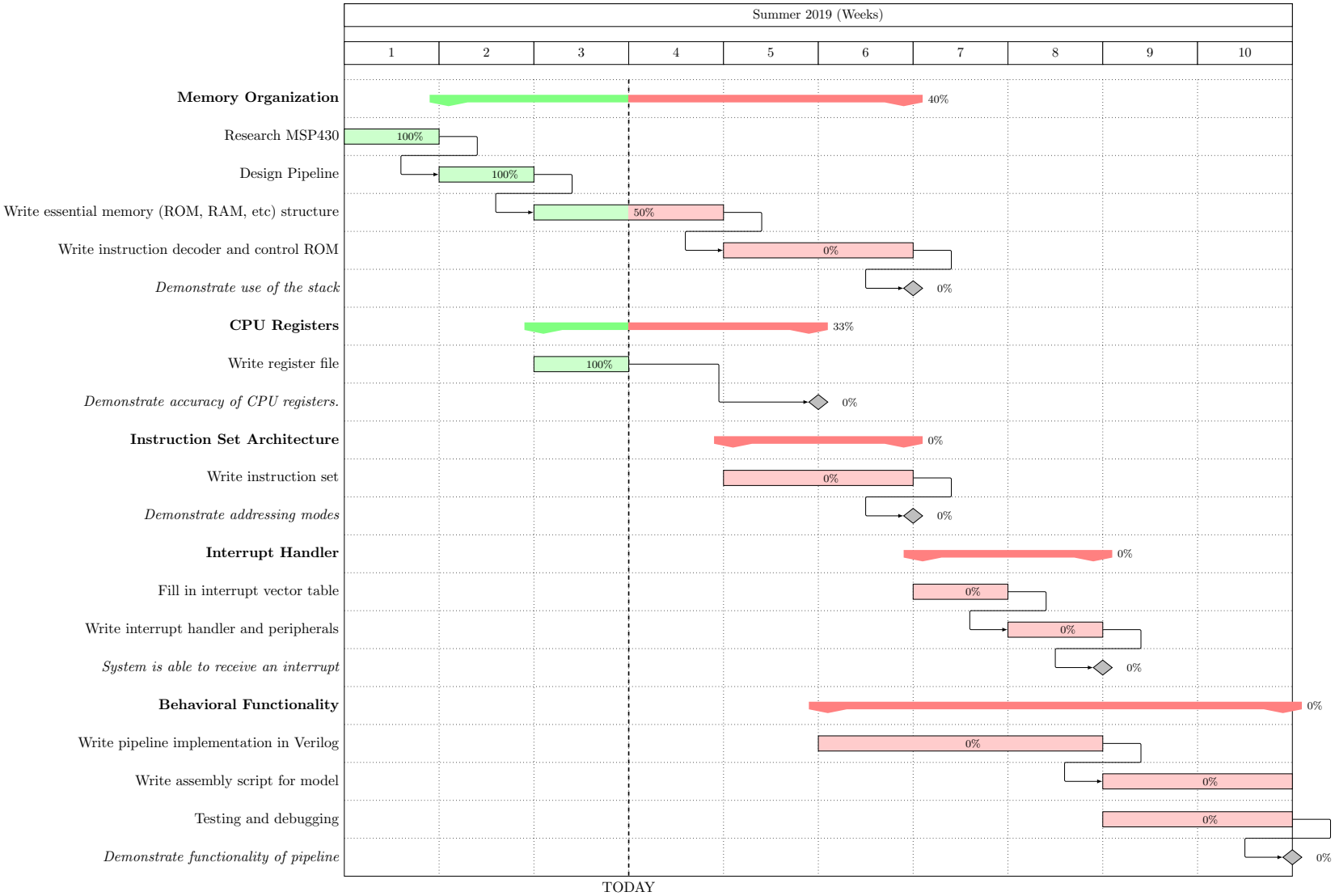
Values Stored in Registers

When `RW` is HIGH, `Din` is stored in R[DA].



Review

Next Week's Deliverable(s)



Next week's deliverable is to write the essential memory structure of the MSP430.

Review

References

- *GPS recalculating*, Image. [Online]. Available: http://culvercitycrossroads.com/wp-content/uploads/2017/11/GPS-Recalulating-1080x675.png.
- *MSP430x2xx Family User's Guide*, Texas Instruments. [Online]. Available: http://www.ti.com/lit/ug/slau144j/slau144j.pdf.
- *MSP430G2x53 Mixed Signal Microcontroller*, Texas Instruments, May 2013. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp430g2553.pdf.
- V. Rodriguez, *Msp430-verilog-model*, Online, Jun. 2019. [Online]. Available: https://github.com/ricerodriguez/msp430-verilog-model.
- *Modeling memories and FSM*, Online, Feb. 2014. [Online]. Available: http://www.asic-world.com/verilog/memory_fsm1.html.