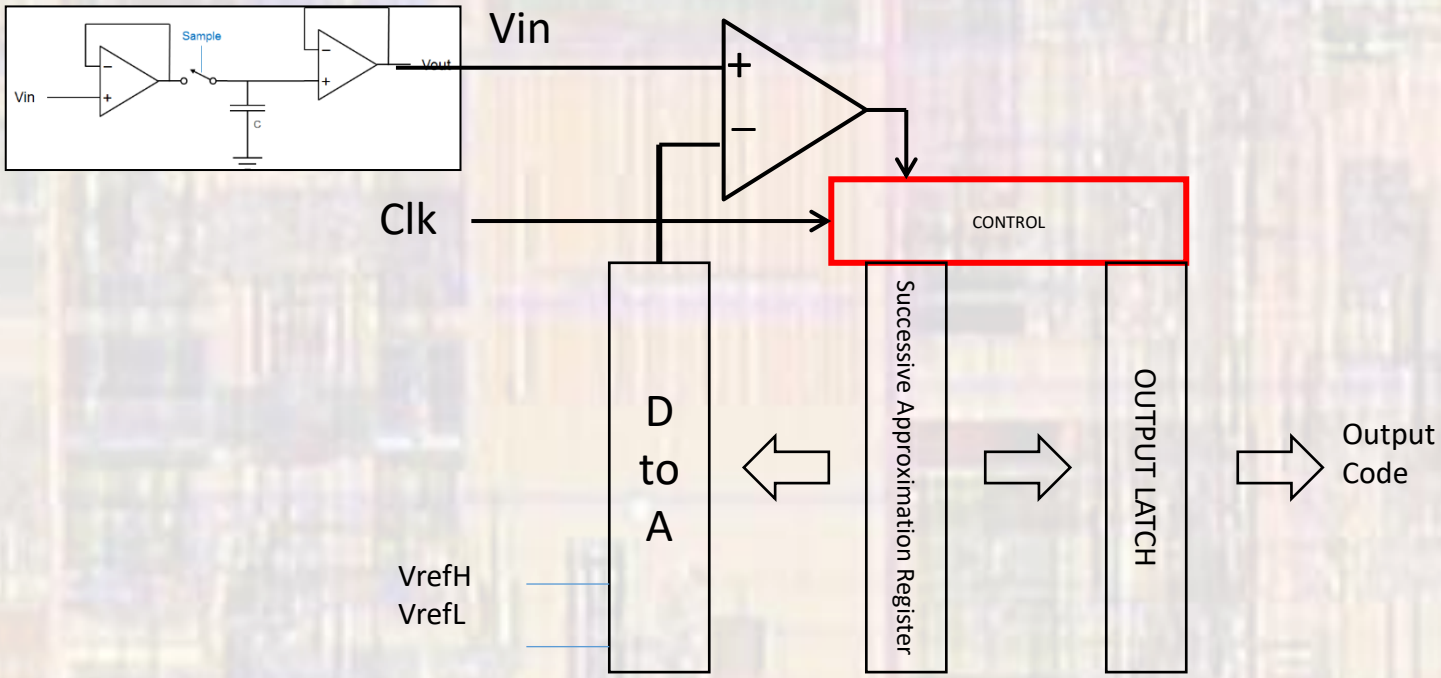
ADC – SAR Controller

Last updated 10/8/26

ADC – SAR Controller

- Successive Approximation A to D

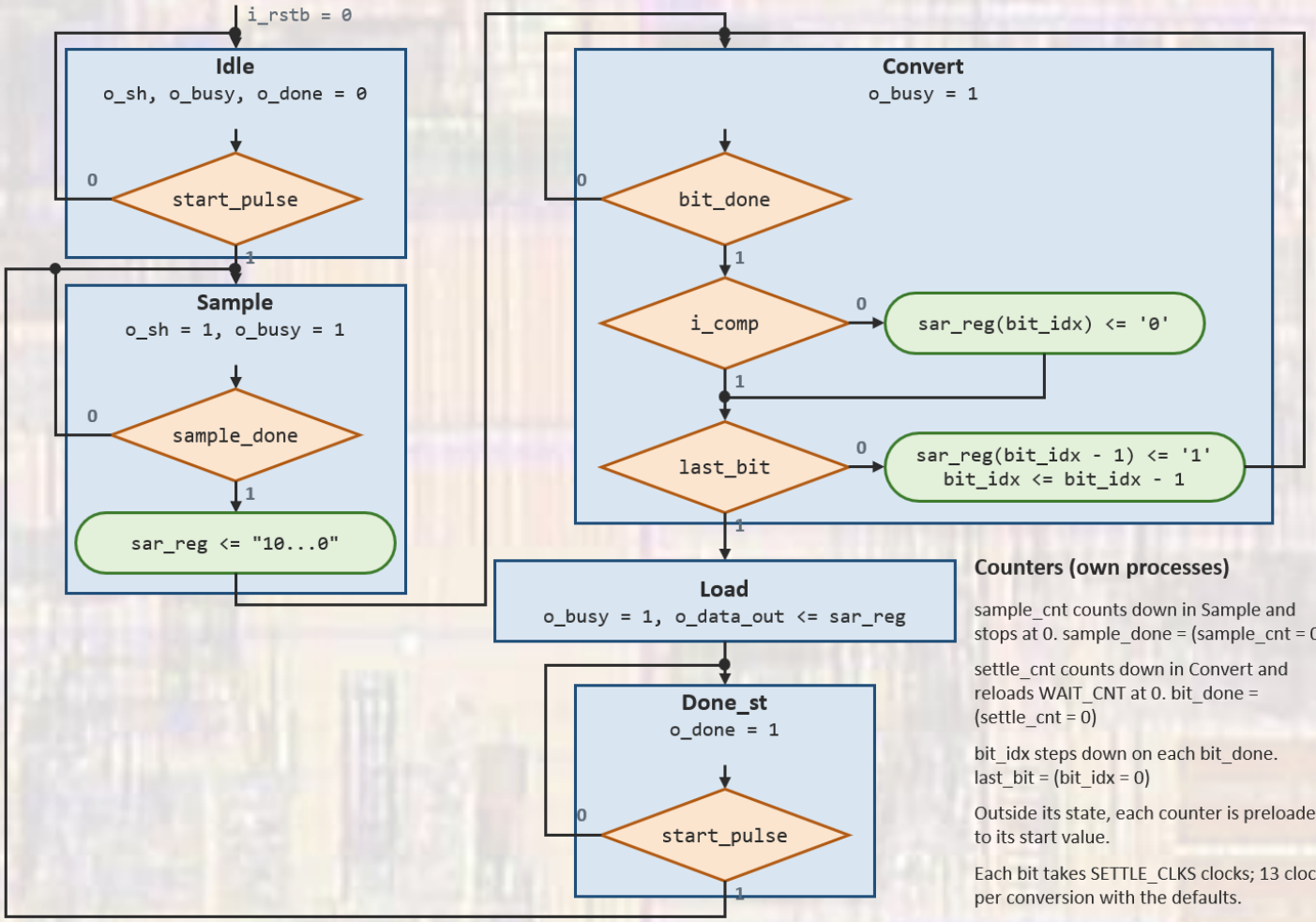


© tj

ADC – SAR Controller

- Controller – ASM Chart

Each state box, with its decisions and conditional boxes, takes one clock. Outputs in a state box are high for the whole state.



Counters (own processes)

sample_cnt counts down in Sample and stops at 0. sample_done = (sample_cnt = 0)

settle_cnt counts down in Convert and reloads WAIT_CNT at 0. bit_done = (settle_cnt = 0)

bit_idx steps down on each bit_done. last_bit = (bit_idx = 0)

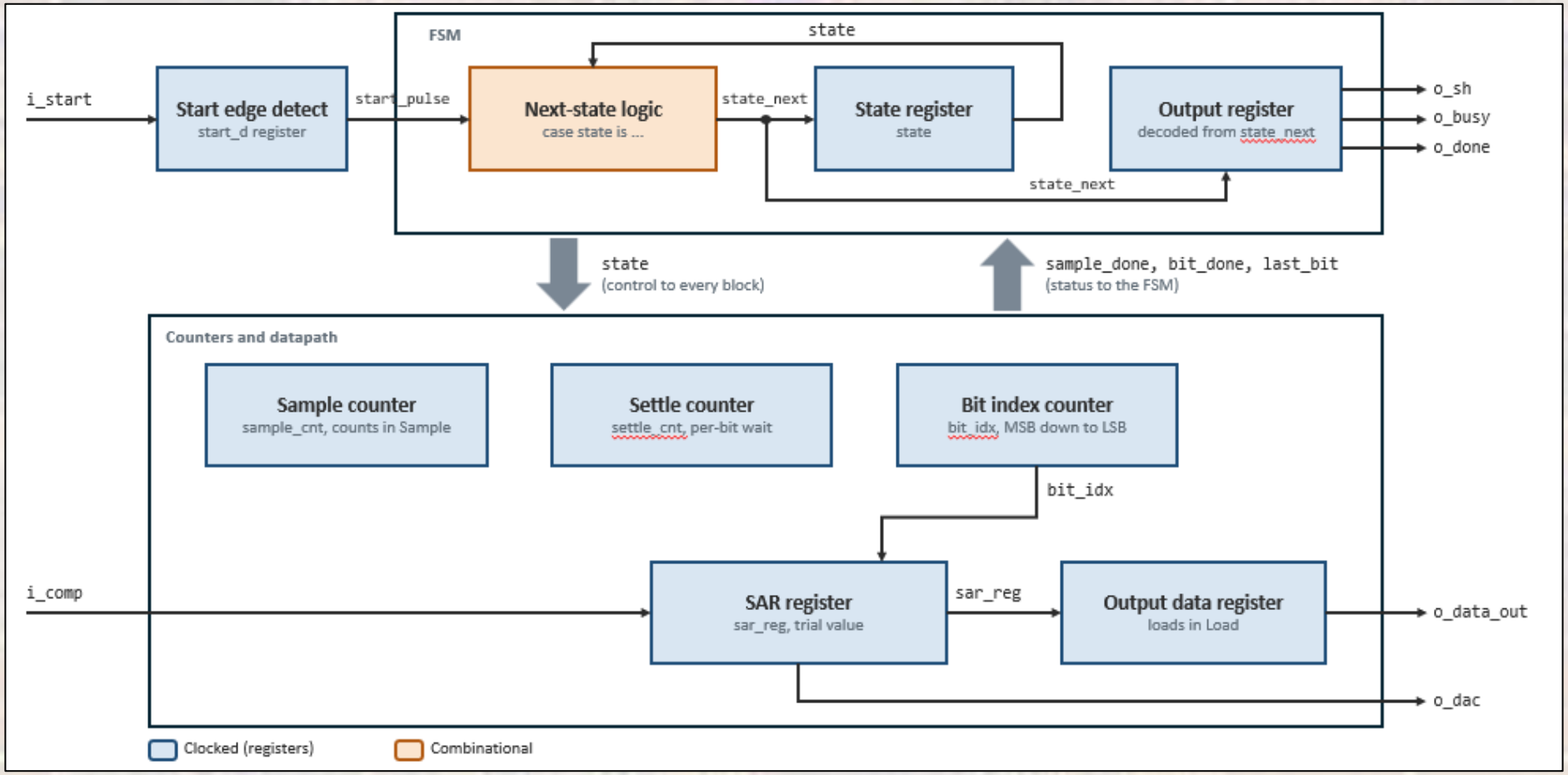
Outside its state, each counter is preloaded to its start value.

Each bit takes SETTLE_CLKS clocks; 13 clocks per conversion with the defaults.

© tj

ADC – SAR Controller

- Controller – FSMD



 © tj

ADC – SAR Controller

- Controller – Description

```
--
-- created 10/6/26
-- tj
--
-- rev 0
----------------------------------------
--
-- Successive Approximation (SAR) ADC controller
--
-- Conversion sequence:
--    Idle    - after reset, wait for a start (rising edge of i_start)
--    Sample  - S/H held in sample (track) mode for SAMPLE_CLKS clocks
--    Convert - S/H in hold, one bit resolved per SETTLE_CLKS clocks, MSB first
--    Load    - final code loaded into the output register
--    Done    - done flag set, wait for the next start
--
-- Conversion time (from start detect to done) is:
--    SAMPLE_CLKS + (N_BITS * SETTLE_CLKS) + 1 clocks
--
----------------------------------------
--
-- Inputs:  clk, rstb, start, comp
-- Outputs: sh, dac, data_out, busy, done
--
-- NOTE: i_comp is expected to be '1' when Vin(held) >= Vdac
--       (bit is kept), '0' when Vin(held) < Vdac (bit is cleared)
-- NOTE: o_sh = '1' -> sample (track),  o_sh = '0' -> hold
-- NOTE: NO input synchronizers
-- NOTE: start is ignored while a conversion is in progress
-- NOTE: o_done stays high until the next start is accepted
-- NOTE: o_sh, o_busy, o_done are registered
--
----------------------------------------
```


© tj

ADC – SAR Controller

- Controller – Entity

```
entity SAR_ADC_ctrl is
    generic(
          N_BITS:         positive := 8;
          SAMPLE_CLKS:    positive := 4;
          SETTLE_CLKS:    positive := 2
          );
    port(
          i_clk:          in  std_logic;
          i_rstb:         in  std_logic;
          i_start:        in  std_logic;
          i_comp:         in  std_logic;

          o_sh:           out std_logic;
          o_sar_to_dac:   out std_logic_vector((N_BITS - 1) downto 0);
          o_data_out:     out std_logic_vector((N_BITS - 1) downto 0);
          o_busy:         out std_logic;
          o_done:         out std_logic
       );
end entity;
```


© tj

ADC – SAR Controller

- Controller – Signals

```
architecture behavioral of SAR_ADC_ctrl is

   ---------------------------------------
   -- helper function - not required
   -- number of bits needed to hold the value N
   function num_bits(N: natural) return positive is
   begin
      if(N < 2) then
         return 1;
      else
         return integer(ceil(log2(real(N + 1))));
      end if;
   end function;
   ---------------------------------------

   -- per bit wait count - decision is made when the counter reaches 0
   constant WAIT_CNT:      natural := SETTLE_CLKS - 1;

   -- counter widths
   constant SAMPLE_WIDTH:  positive := num_bits(SAMPLE_CLKS);
   constant SETTLE_WIDTH:  positive := num_bits(SETTLE_CLKS);
   constant IDX_WIDTH:     positive := num_bits(N_BITS);

   -- controller states
   type state_type is (Idle, Sample, Convert, Load, Done);
   signal state:        state_type;
   signal state_next:   state_type;

   -- start edge detect
   signal start_det:    std_logic;
   signal start_pulse:  std_logic;

   -- sample_cnt - S/H sample timer
   -- settle_cnt - per bit DAC / comparator settle timer
   -- bit_idx    - bit currently being tested
   -- sar_reg    - successive approximation register (trial value sent to the DAC)
   signal sample_cnt:   unsigned((SAMPLE_WIDTH - 1) downto 0);
   signal settle_cnt:   unsigned((SETTLE_WIDTH - 1) downto 0);
   signal bit_idx:      unsigned((IDX_WIDTH - 1) downto 0);
   signal sar_reg:      std_logic_vector((N_BITS - 1) downto 0);

   -- counter status, not required but make the code more readable
   signal sample_done:  std_logic;
   signal bit_done:     std_logic;
   signal last_bit:     std_logic;
```


© tj

ADC – SAR Controller

- Controller FSM – Start Edge Detect

```
-- start edge detect - one conversion per start, even if start is held high
process(i_clk, i_rstb)
begin
   if(i_rstb = '0') then
      start_det <= '0';
   elsif(rising_edge(i_clk)) then
      start_det <= i_start;
   end if;
end process;

start_pulse <= i_start and not start_det; -- rising edge of start
```


© tj

ADC – SAR Controller

- Controller FSM – Next State Logic

```
-- next state logic
process(state, start_pulse, sample_done, bit_done, last_bit)
begin
   case state is
      when Idle =>
         if(start_pulse = '1') then
            state_next <= Sample;
         else
            state_next <= state;
         end if;
      when Sample =>
         if(sample_done = '1') then
            state_next <= Convert;
         else
            state_next <= state;
         end if;
      when Convert =>
         if((bit_done = '1') and (last_bit = '1')) then
            state_next <= Load;
         else
            state_next <= state;
         end if;
      when Load =>
         state_next <= Done;
      when Done =>
         if(start_pulse = '1') then
            state_next <= Sample;
         else
            state_next <= state;
         end if;
      when others =>
         state_next <= Idle;
   end case;
end process;
```


© tj

ADC – SAR Controller

- Controller FSM – Register Logic

```
-- state register
process(i_clk, i_rstb)
begin
   if(i_rstb = '0') then
      state <= Idle;
   elsif(rising_edge(i_clk)) then
      state <= state_next;
   end if;
end process;
```


© tj

ADC – SAR Controller

- Controller FSM – Output Logic

```
   -- outputs - registered
   process(i_clk, i_rstb)
   begin
      if(i_rstb = '0') then
         o_sh     <= '0';     -- hold
         o_busy   <= '0';
         o_done   <= '0';
      elsif(rising_edge(i_clk)) then
         case state_next is
            when Sample =>
               o_sh     <= '1';
               o_busy   <= '1';
               o_done   <= '0';
            when Convert =>
               o_sh     <= '0';
               o_busy   <= '1';
               o_done   <= '0';
            when Load =>
               o_sh     <= '0';
               o_busy   <= '1';
               o_done   <= '0';
            when Done =>
               o_sh     <= '0';
               o_busy   <= '0';
               o_done   <= '1';
            when others =>
               o_sh     <= '0';
               o_busy   <= '0';
               o_done   <= '0';
         end case;
      end if;
   end process;
```


© tj

ADC – SAR Controller

- Controller Data Path – S/H Counter

```
-- sample (S/H) counter - preloaded outside Sample, counts down in Sample
process(i_clk, i_rstb)
begin
   if(i_rstb = '0') then
      sample_cnt <= to_unsigned(SAMPLE_CLKS - 1, SAMPLE_WIDTH);
      sample_done <= '0';
   elsif(rising_edge(i_clk)) then
      if(state = Sample) then
         if(sample_cnt /= 0) then
            sample_cnt <= sample_cnt - 1;
            sample_done <= '0';
         else
            sample_cnt <= sample_cnt;
            sample_done <= '1';
         end if;
      else
         sample_cnt <= to_unsigned(SAMPLE_CLKS - 1, SAMPLE_WIDTH);
         sample_done <= '0';
      end if;
   end if;
end process;
```


© tj

ADC – SAR Controller

- Controller Data Path – DAC Settling Counter

```
-- settle counter - preloaded outside Convert, reloads after each bit decision
process(i_clk, i_rstb)
begin
   if(i_rstb = '0') then
      settle_cnt <= to_unsigned(WAIT_CNT, SETTLE_WIDTH);
      bit_done   <= '0';
   elsif(rising_edge(i_clk)) then
      if((state = Convert) and (settle_cnt /= 0)) then
         settle_cnt <= settle_cnt - 1;
         bit_done   <= '0';
      else
         settle_cnt <= to_unsigned(WAIT_CNT, SETTLE_WIDTH);
         bit_done   <= '1';
      end if;
   end if;
end process;
```


© tj

ADC – SAR Controller

- Controller Data Path – Bit Index Counter

```
-- bit index counter - MSB outside Convert, steps down after each bit decision
process(i_clk, i_rstb)
begin
   if(i_rstb = '0') then
      bit_idx <= to_unsigned(N_BITS - 1, IDX_WIDTH);
   elsif(rising_edge(i_clk)) then
      if(state /= Convert) then
         bit_idx <= to_unsigned(N_BITS - 1, IDX_WIDTH);
      else
         if((bit_done = '1') and (last_bit = '0')) then
            bit_idx <= bit_idx - 1;
         else
            bit_idx <= bit_idx;
         end if;
      end if;
   end if;
end process;
-- pulse the last bit signal
last_bit    <= '1' when (bit_idx = 0) else '0';
```


© tj

ADC – SAR Controller

- Controller Data Path – SAR

```
-- successive approximation register
process(i_clk, i_rstb)
begin
   if(i_rstb = '0') then
      sar_reg <= (others => '0');
   elsif(rising_edge(i_clk)) then
      if((state = Sample) and (sample_done = '1')) then
         sar_reg              <= (others => '0');
         sar_reg(N_BITS - 1)  <= '1';                    -- first trial - MSB
      elsif(bit_done = '1') then
         if(i_comp = '0') then
            sar_reg(to_integer(bit_idx)) <= '0';         -- Vin < Vdac - trial too big
         end if;
         if(last_bit = '0') then
            sar_reg(to_integer(bit_idx - 1)) <= '1';     -- next trial bit
         end if;
      end if;
   end if;
end process;
```


© tj

ADC – SAR Controller

- Controller Data Path – Data Output Register

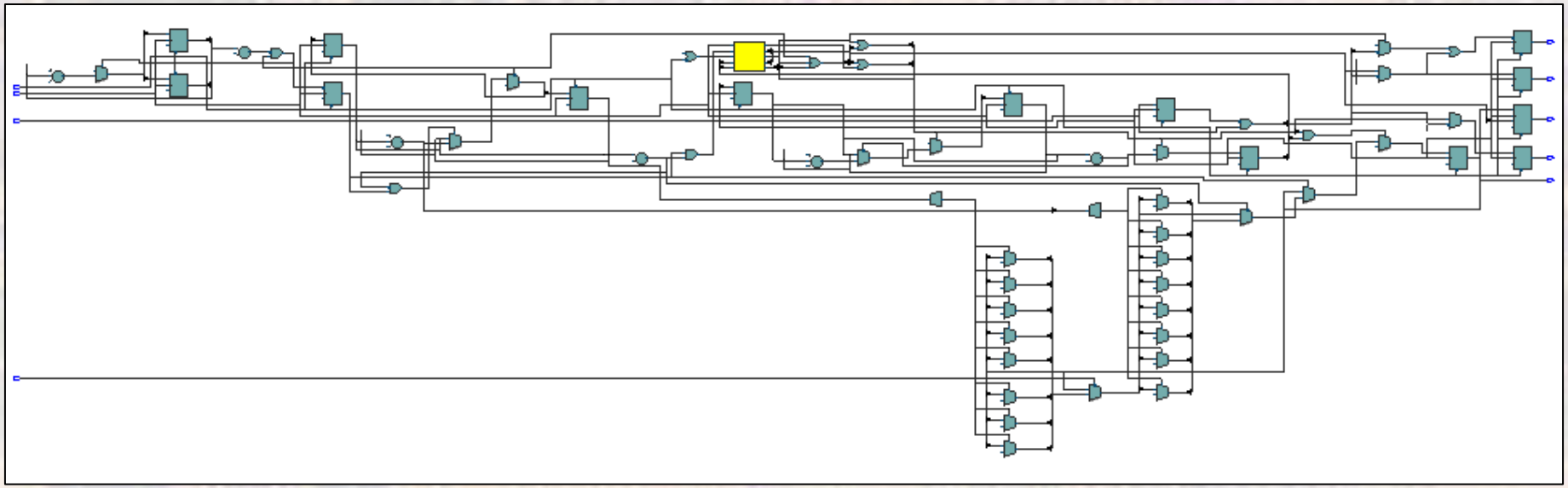
```
-- output data register
process(i_clk, i_rstb)
begin
   if(i_rstb = '0') then
      o_data_out <= (others => '0');      -- predictable output in reset
   elsif(rising_edge(i_clk)) then
      if(state = Load) then
         o_data_out <= sar_reg;
      else
         o_data_out <= o_data_out;
      end if;
   end if;
end process;

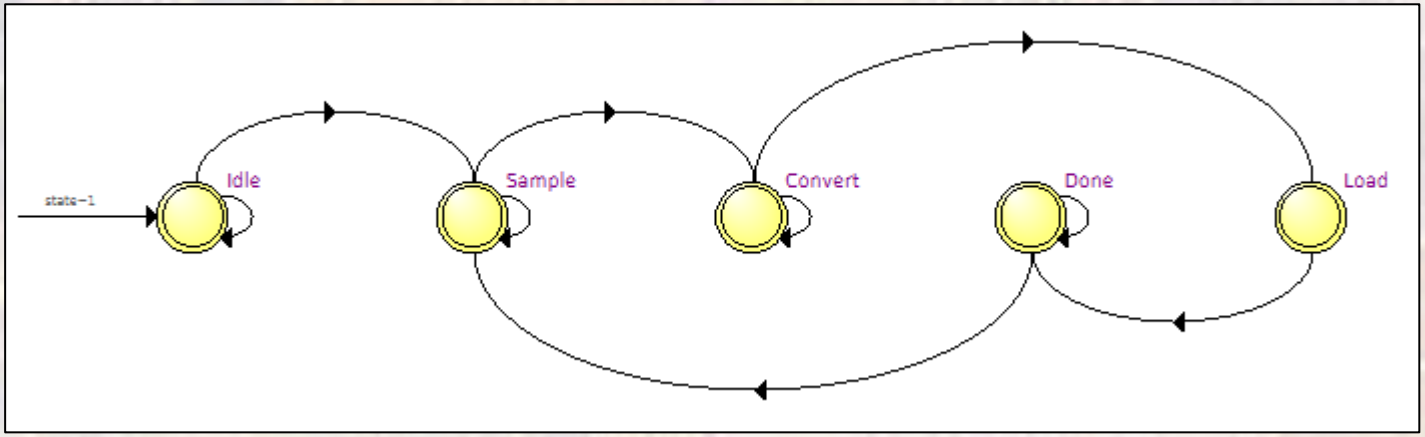
o_sar_to_dac <= sar_reg;
```


© tj

ADC – SAR Controller

- Controller RTL





© tj