Design of Control and Status Register (CSR) with APB interface

Project ID: 26441

B.Tech. Project Report
submitted for fulfillment of
the requirements for the
Degree of Bachelor of Technology
Under Biju Pattnaik University of Technology

Submitted By

Deepika Adhikary	**ROLL NO. ECE201911579**
Sai Swarup Patnaik	**ROLL NO. ECE201910057**

2022 – 2023

Under the guidance of

Prof. Rajesh Kumar Dash

NIST INSTITUTE OF SCIENCE & TECHNOLOGY (Autonomous)
Institute Park, Palur Hills, Berhampur, Odisha – 761008, India

ABSTRACT

This report deals with the design and implementation of the Control Status Register (CSR) for SoCs and presents the proposed architecture of the CSR along with brief details of individual blocks. The control register stores different digital signals for various commands for components inside the SoCs like read and write operations, and updates the status register when read from or written into registers. The timing parameters are written by the APB (Advanced Peripheral Bus) master and CSR receives this information through the APB slave interface which is at the controller side and stores this information in respective registers of CSR. All components are connected through the register, which stores all the information (like address, and data). Finally, we synthesize, place, and route the entire design to create IP (Intellectual property) by implementing APB (Advanced Peripheral Bus).

The entire design was coded in Verilog hardware description language and simulated using Xilinx 2018.2 version. The results of various operations were verified using different test cases.

Keywords: APB Protocol, CSR, Digital Design.

ACKNOWLEDGEMENT

It is our proud privilege to epitomize my deepest sense of gratitude and indebtedness to our advisor, **Prof. Rajesh Kumar Dash** for his valuable guidance, keen and sustained interest, intuitive ideas, and persistent endeavour. His inspiring assistance, laconic reciprocation, and affectionate care enabled me to complete our work smoothly and successfully.

We extend our sincere thanks to **Prof. Rajesh Kumar Dash, B.Tech. Project Coordinator and Prof. Bibhuti Bhusan Mishra, Department Project Coordinator**, for giving us the opportunity and motivating us to complete the Minor Project within the stipulated period and providing a helping environment.

We would also like to thank **Dr. Sukant K. Mohapatra (Chairman, NIST) and Dr. M. Suresh (HoD, Department of Electronics & Communication Engg.)** for having been the ultimate source of inspiration and moral support.

We would also like to extend our heartfelt gratitude to our parents and friends for their support and help.

Deepika Adhikary

Sai Swarup Patnaik

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

A system-on-chip (SoC) is an integrated circuit (IC) that integrates all components of a computer or other electronic system into a single chip. It may contain digital, analog, radio-frequency functions, and mixed-signal all on a single chip substrate. If we want to control or know the status of any component inside a SoC then two things we can do, by using the external pin through which we can control or see the status of it. But it increases the cost of SoC as pins are very costly and eat up many areas. So, we proposed an approach where we use the Control and Status Register by which we can Read/write/control the components inside the SoC using only a few pins.

2. DESIGN METHODOLOGY

- First, we implementation the RW(Read-Write) and RO (Read Only) register.
- 8-bit Reading and Writing register design using D-FF which is the simplest of all Flip-Flops.
- Design the architecture of Control and status register by connect the 8-bit RW register with address decoder and 8-bit mux.
- Decoder is used to select a particular RW register, and 8-bit mux is connected to read a particular data from a particular register.
- Then implement the APB protocol as an interface to CSR.
- Finally, we Synthesis, place and route the design.

3. REGISTER

- When we write our programs, we take for granted the fact that we can declare a variable x and assign it some value and be assured that it will retain said value until we specify it is to be modified.
- This sets us a digital design task:
  1. How do we connect our logic gates in such a way that the resulting circuit will store a value?
- A register is a group of binary cells suitable for holding binary information. A group of cascaded flip-flops used to store related bits of information is known as a register.
- Two types of registers we are using here
  2. Read-Write Register
  3. Read-Only Register
- Flip-flop is a 1-bit memory cell which can be used for storing the digital data. To increase the storage capacity in terms of number of bits, we must use a group of flip-flops. Such a group of flip-flops is known as a **Register**. The **n-bit register** will consist of **n** number of flip-flop, and it can store an **n-bit** word.
- Most commonly we use D flip-flop as a storage element where there is a need to store one-bit could be 0 or 1. It is one of the simplest of all types of flip-flops. It is also called as the Data flip-flop or Delay flip-flop, figure 1 shows the basic architecture of D-FF.

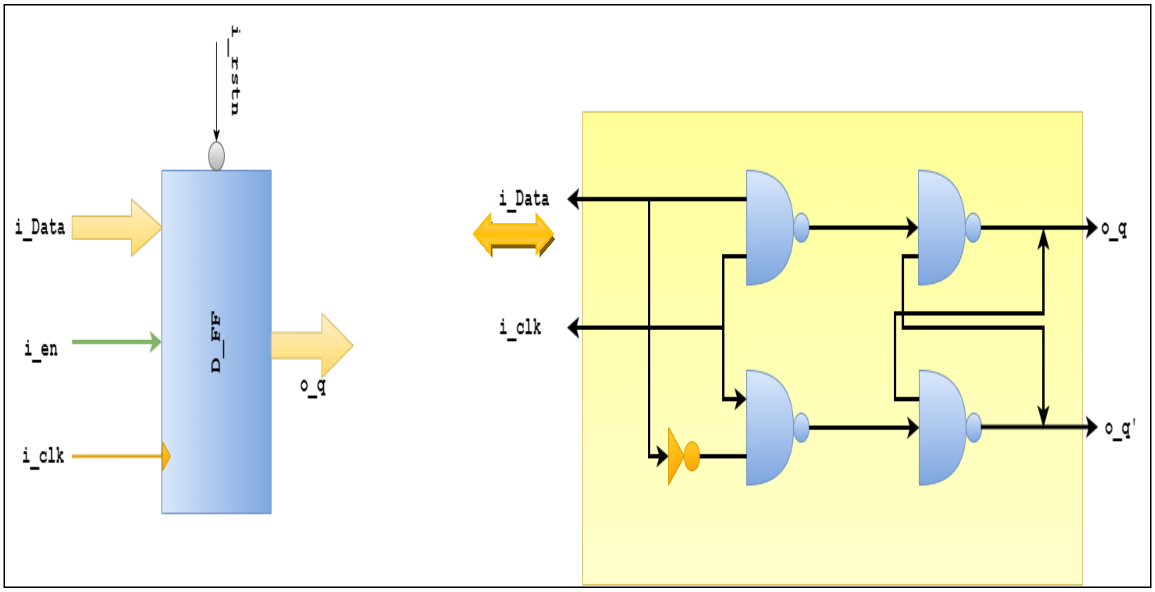

Figure 3.1: Schematic of Read-Write Register

- At the **positive edge** of the **clock** if **reset input** is **low**, then **output o_q** automatically will be **reset (Logic 0).** Else, the **output o_q** will be **same** as the **i_Data** input.
- We want the register to be able to update a value, but we do not want it automatically updating every single clock cycle. We want to use our instructions to control exactly when the value updates. What we need is a **control signal** to help here. We call it a **enable signal** and denote it **en** and the pin is called an **enable pin** and is denoted by **i_en**. When **i_en = 1**, we want to allow the device **to update**. When **i_en = 0,** we want it **to hold** its value. So, we can build a bit of a truth table to figure out the logic necessary to make this happen. See Table 3.1.

Table 3.1: Truth table for the D Flip-Flop with Active-low reset, enable and clock.

i_rstn	***i_clk***	***i_en***	***o_q(n+1)***
0	_	_	0
1	0	_	o_q
1	1	_	o_q
1	↑	0	o_q
1	↑	1	o_q

- When we Put all this one-bit D-FF together, they finally build a register!
- We want to be able to store large 32- and 64-bit numbers, not just a single boring bit. To do this, we need to network together several one-bit storage D-FF.
- So, the design of an 8-bit register using D-FF to store the state looks like Figure 2.

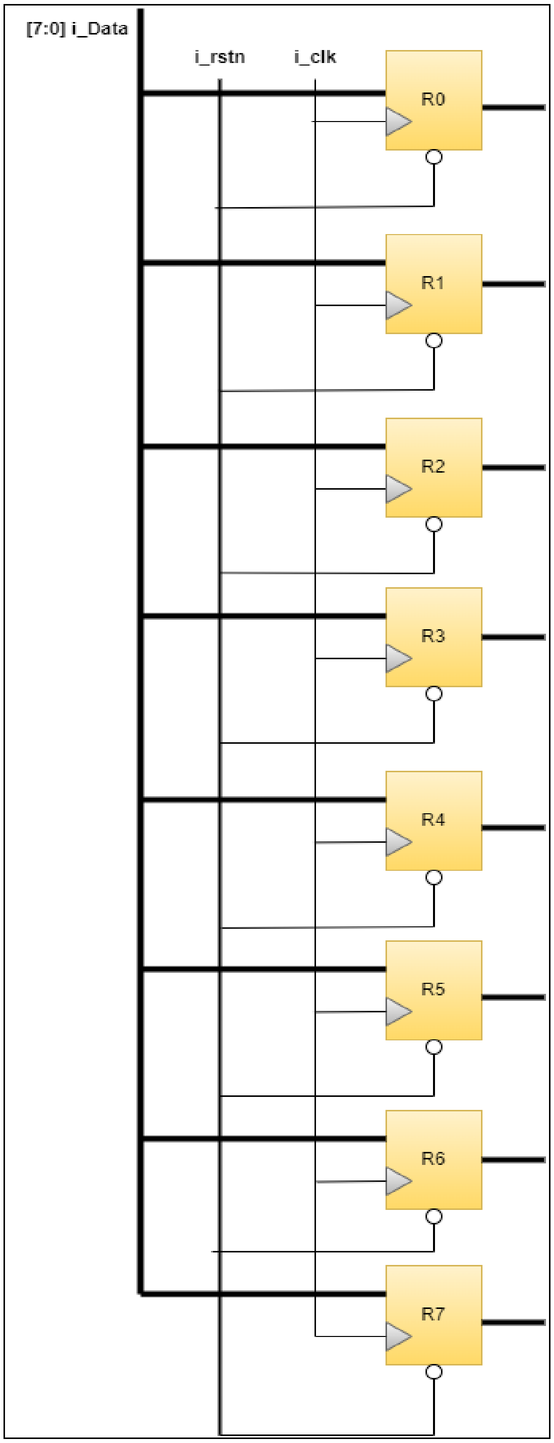

Figure 3.2: 8-bit register schematics

4. ARCHITECTURE OF CSR

- The multiplexer lets us select one of several inputs to pass through to the next device. The decoder allows us to select one of several connected devices to activate in response to an input. One of the most important devices to be driven by a decoder is a register.
- The read-write contains Decoder, D flip-flop and MUX.
- The 3:8 decoder (pronounced "3-to-8 decoder"). It takes a three-bit input and activates one of the 8 output lines in response. This is an active-high decoder so when the output line is activated it is logic-1 and the inactive lines are logic-0. You can also have an active-low decoder symbolized using inversion bubbles on each of the outputs.
- Each output line is attached to the enable signal(i_en) for one of the Read-Write register as shown in figure 4.1. In this way we can replace the 8 individual enable signals (i_en) for each of the RW registers with a single 3-bit input (i_addr).

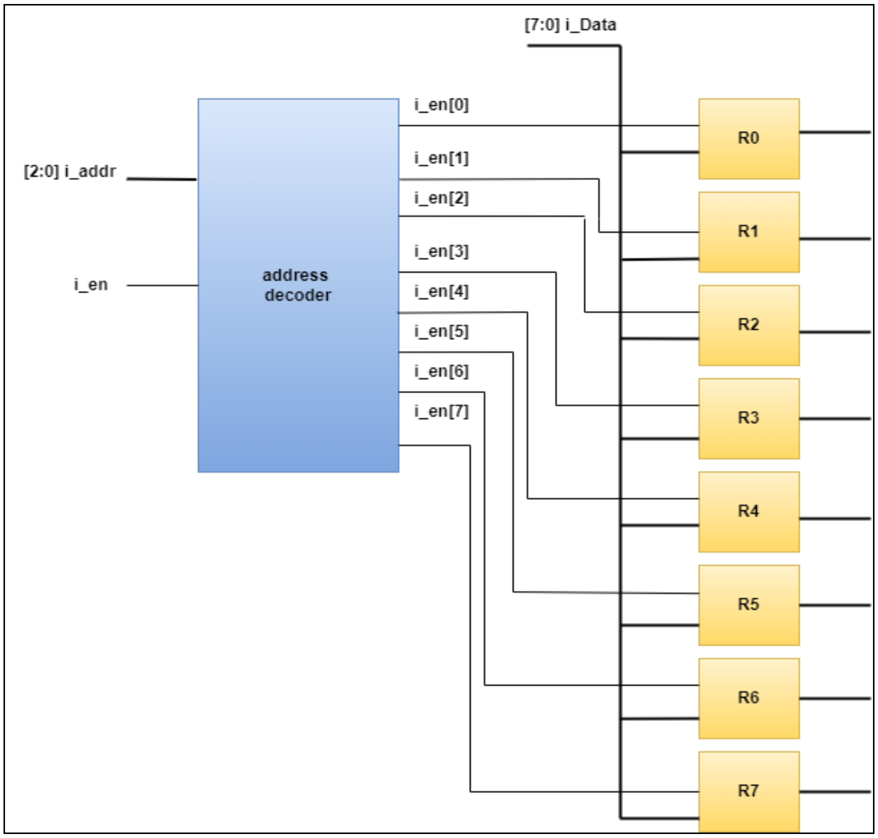

Figure 4.1: Connection diagram of address decoder with 8-bit RW register

There are still two problems: -

1. Since one output line of the decoder must always be active, we are forced to update one of the registers each clock cycle.
2. How do we deal with the 8 separate outputs from all these registers?

Solution:

1. We can solve the first problem by adding a control signal to the decoder to **enable** or **disable** it. When the **decoder is disabled** then **no output line is active**. Therefore, we will not be required to change the value in a register in each clock cycle.
2. We can solve the second problem by using **a single 8:1 MUX** which can **select one of the registers** to output from the register file as shown in figure 4.2.

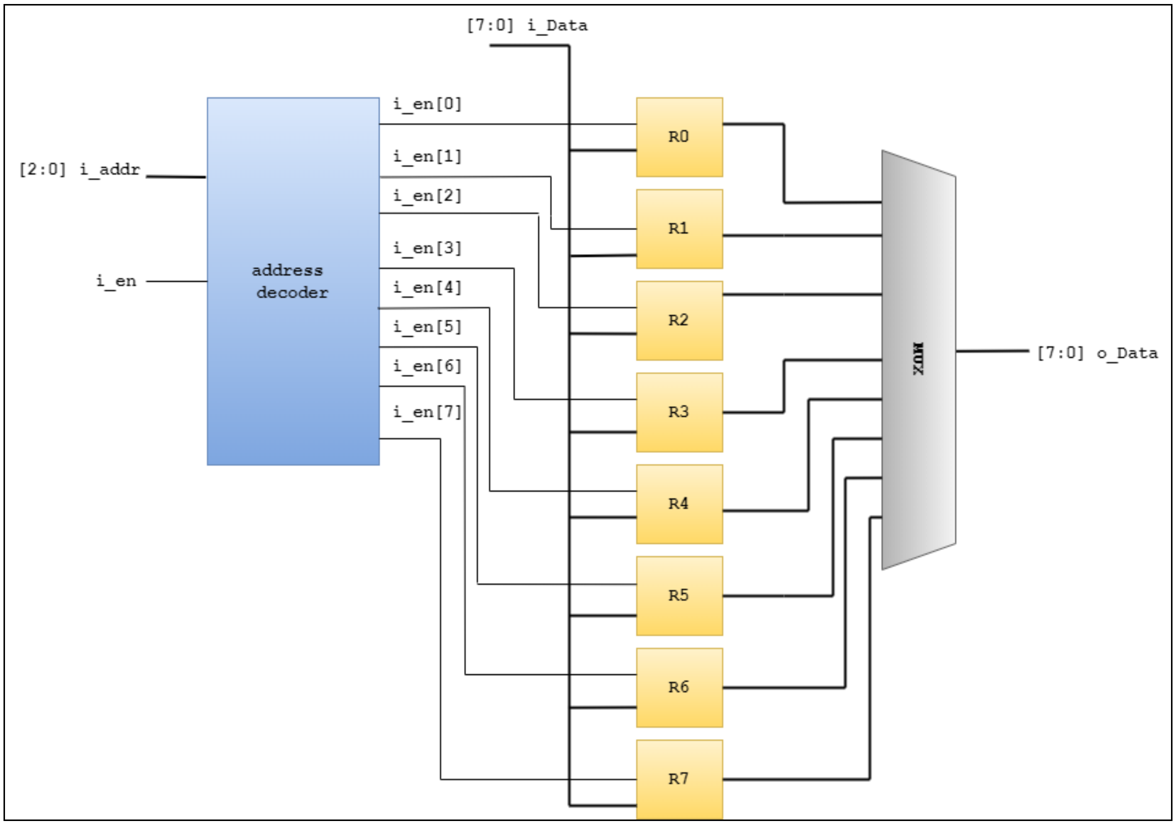

Figure 4.2: Connection diagram of CSR Architecture

5. APB PROTOCOL

5.1 What is APB Protocol

An Advanced Peripheral Bus (APB) is a part of the Advanced Microcontroller Bus Architecture (AMBA). In addition, it specifies a low-cost interface with low power consumption and reduced interface complexity. As APB protocol is not pipelined, it can be used to connect peripherals that do not need the high performance of the AXI protocol. It simplifies the integration of APB peripherals into any design flow by tying signal transitions to the rising edge of the clock. At least two cycles are required to complete a transfer.

5.2 APB Signals

Table 5.1: APB Signals

Signal	Source	Description
PCLK	Clock source	Clock. The rising edge of PCLK times all transfers on the APB.
PRESETn	System bus equivalent	Reset. The APB reset signal is active LOW. This signal is normally connected directly to the system bus reset signal
PSELx	APB bridge	Select. The APB bridge unit generates this signal to each peripheral bus slave. It indicates that the slave device is selected and that a data transfer is required. There is a **PSELx** signal for each slave.
PADDR	APB bridge	Address. This is the APB address bus. It can be up to 32 bits wide and is driven by the peripheral bus bridge unit.
PENABLE	APB bridge	Enable. This signal indicates the second and subsequent cycles of an APB transfer
PWRITE	APB bridge	Direction. This signal indicates an APB write access

		when HIGH and an APB read access when LOW
PWDATA	APB bridge	Write data. This bus is driven by the peripheral bus bridge unit during write cycles when **PWRITE** is HIGH. This bus can be up to 32 bits wide.
PREADY	Slave interface	Ready. The slave uses this signal to extend an APB transfer
PRDATA	Slave interface	Read Data. The selected slave drives this bus during read cycles when **PWRITE** is LOW. This bus can be up to 32-bits wide.

5.3 APB Transfer Conditions

There are two types of transfer conditions in the APB interface:

- **Write Transfer.**
- **Read Transfer.**

5.3.1 Write Transfer

This section describes the following types of write transfer:

- **With no wait states.**
- **With wait states.**

With no wait States:

- When **PCLK** rises at **T1**, a write transfer starts with address **PADDR**, write data **PWDATA**, write signal **PWRITE**, and select signal **PSEL.** This is called the Setup phase of the write transfer.
- As **PCLK** rises at **T2**, enable signal **PENABLE** and ready signal **PREADY** will be detected.

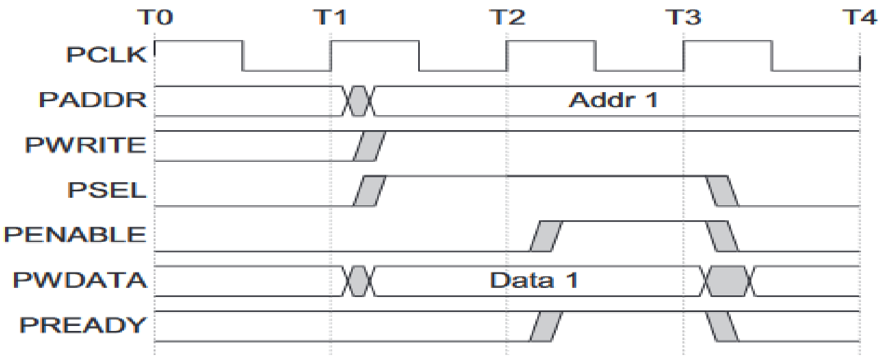

Figure 5.1: Write-transfer with no wait state

- As soon as **PENABLE** is asserted, the Access phase begins.
- When asserted, **PREADY** indicates that the slave is ready to complete the transfer when **PCLK** next rises.
- At **T3**, at the end of the access phase, all address **PADDRs**, write data **PWDATA**s, and control signals remain valid.
- At the end of the transfer, the enable signal **PENABLE** is de-asserted. **PSEL**, the select signal, is also de-asserted unless another transfer to the same peripheral follows immediately after the first.

With wait state:

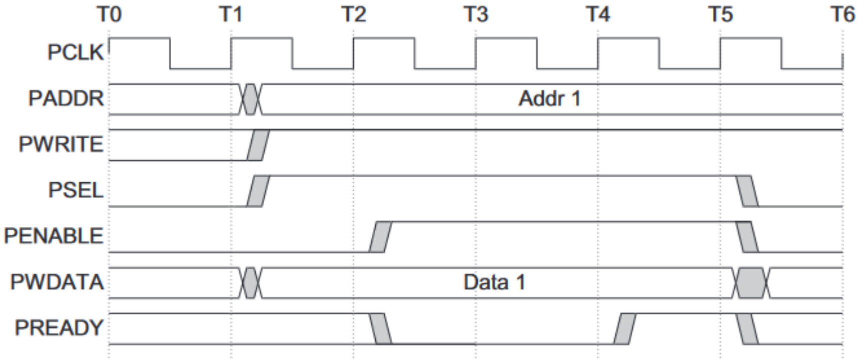

Figure 5.2: Write transfer with wait state

- In **Figure 5.2**, the slave shows how to extend the transfer by using the **PREADY** signal. During an Access phase, the slave extends the transfer by driving **PREADY LOW** when **PENABLE** is **HIGH**.

- The following signals remain unchanged while **PREADY** remains LOW:
  - → Address, **PADDR.**
  - → Write signal, **PWRITE.**
  - → Select signal, **PSEL.**
  - → Enable signal, **PENABLE.**
  - → Write data, **PWDATA.**
  - → Write strobes, **PSTRB.**
- When **PENABLE** is **LOW**, **PREADY** can take any value. This ensures that peripherals with fixed two-cycle access can therefore be paired with **PREADY HIGH**.

5.3.2 Read Transfer

Two types of read transfer are described in this section:

- **With no wait states.**
- **With wait states.**

With no wait states:

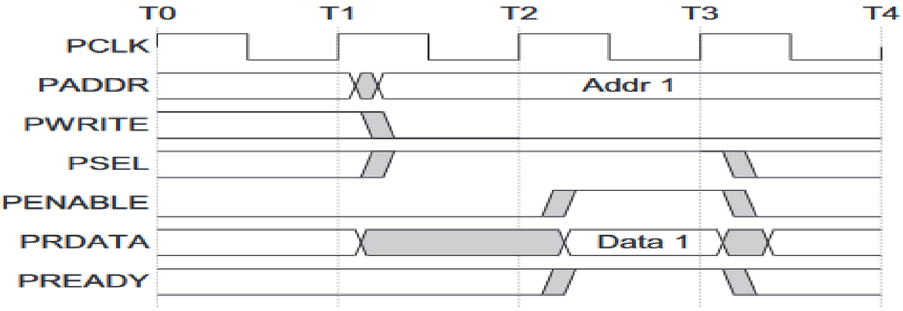

Figure 5.3: Read Transfer with no wait state.

Figure 5.3 shows a read transfer. The timing of the address, write, select, and enable signals are as described in Write transfers. The slave must provide the data before the end of the read transfer.

With wait state:

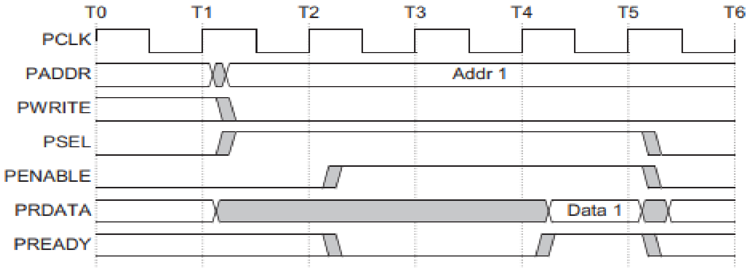

Figure 5.4: Read Transfer with wait state.

Figure 5.4 shows how the **PREADY** signal can extend the transfer. The transfer is extended if **PREADY** is driven LOW during an Access phase.

The protocol ensures that the following remain unchanged for the additional cycles:

- address, **PADDR**.
- write signal, **PWRITE**
- select signal, **PSEL**
- enable signal, **PENABLE**

Fig. 5.4 shows that two cycles are added using the **PREADY** signal. However, you can add any number of additional cycles, from zero upwards.

6. OPERATING STATE

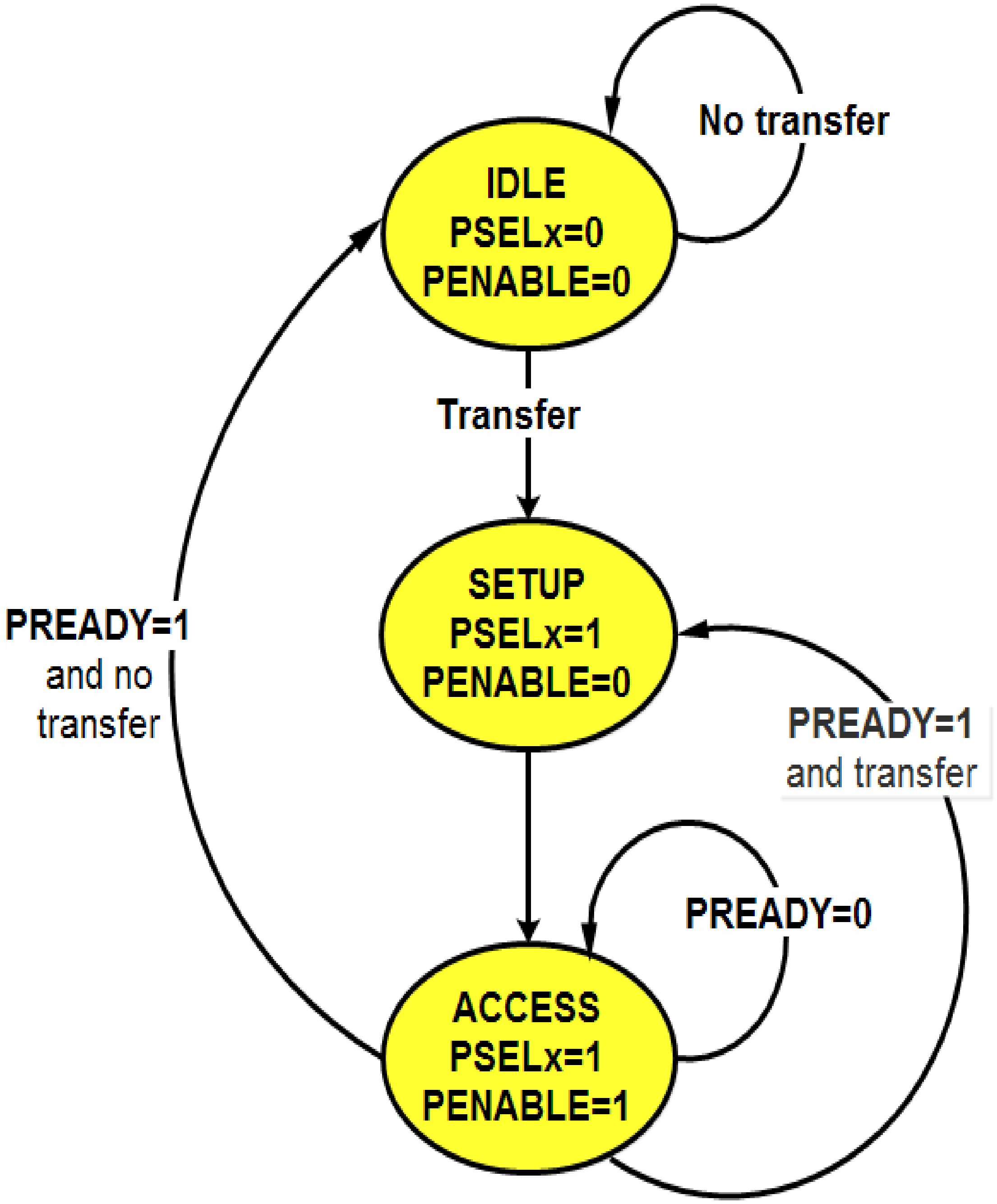

Figure 6.1: State Machine.

The state machine operates through the following states:

- **IDLE:** This is the default state of the **APB**.
- **SETUP:** When a transfer is required the bus moves into the **SETUP** state, where the appropriate select signal, **PSELx,** is asserted. The bus only remains in the **SETUP** state for one clock cycle and always moves to the **ACCESS** state on the next rising edge of the clock.
- **ACCESS:** The enable signal, PENABLE, is asserted in the **ACCESS** state. The address, write, select, and write data signals must remain stable during the transition from the **SETUP** to **ACCESS** state. Exit from the **ACCESS** state is controlled by the **PREADY** signal from the slave: If **PREADY** is held **LOW** by the slave then the peripheral bus remains in the **ACCESS** state.

7. CSR ARCHITECTURE

7.1 Schematic

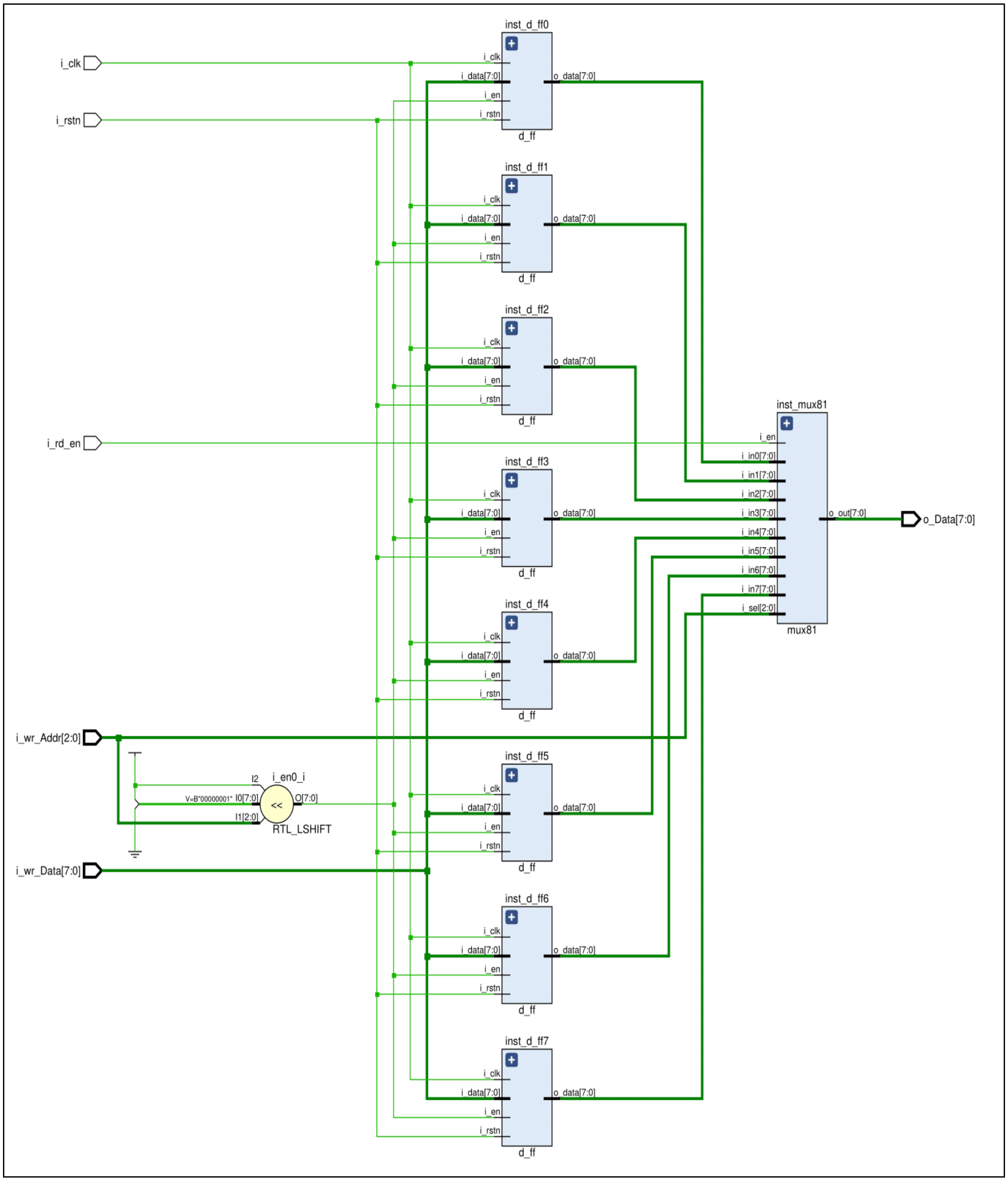

Figure 7.1: Schematic of CSR

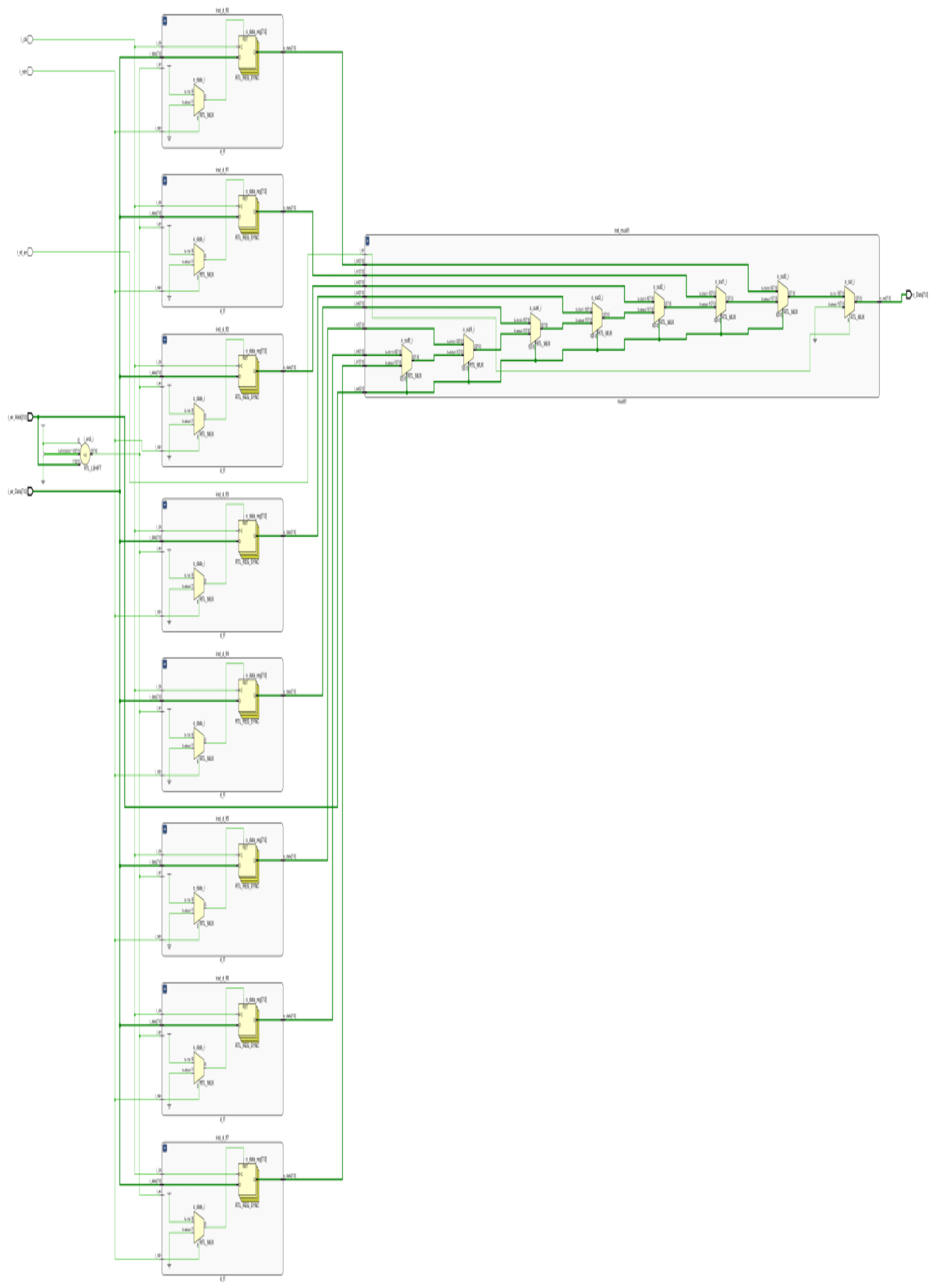

Figure 7.2: Elaborate Schematic

7.2 Simulation Waveform

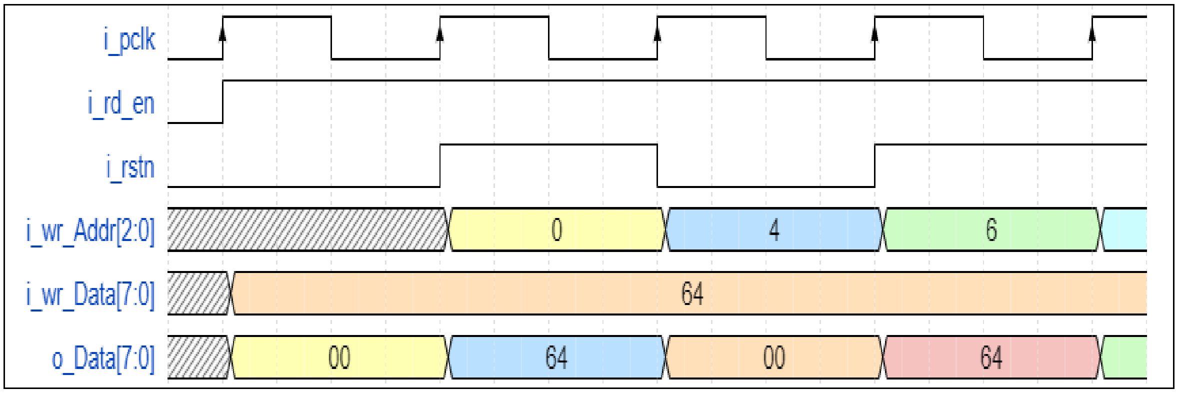

Figure 7.3: Simulation waveform of CSR

8. APB STATE MACHINE

8.1 Schematic

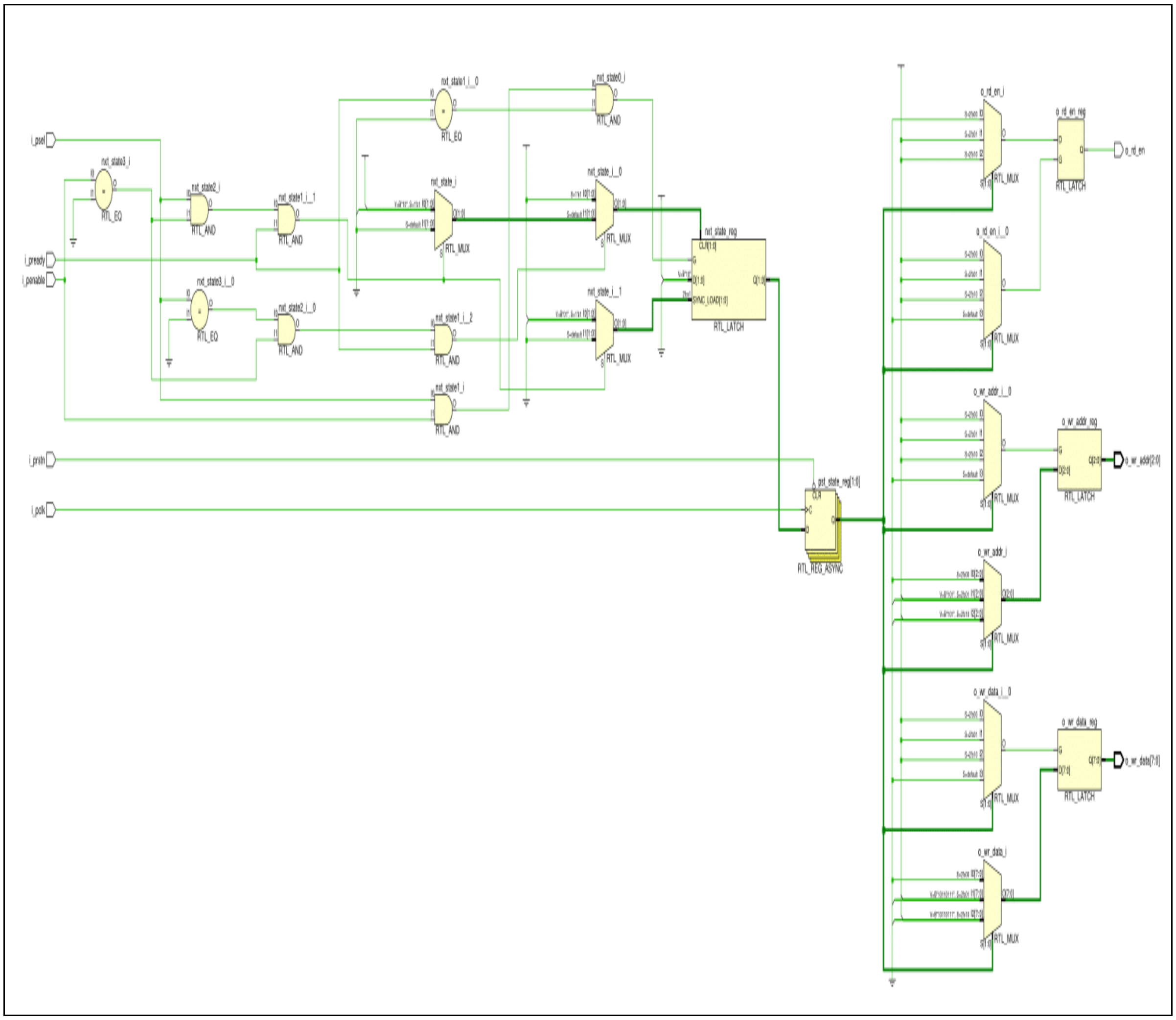

Figure 8.1: Elaborate Schematic of APB

8.2 Simulation Waveform

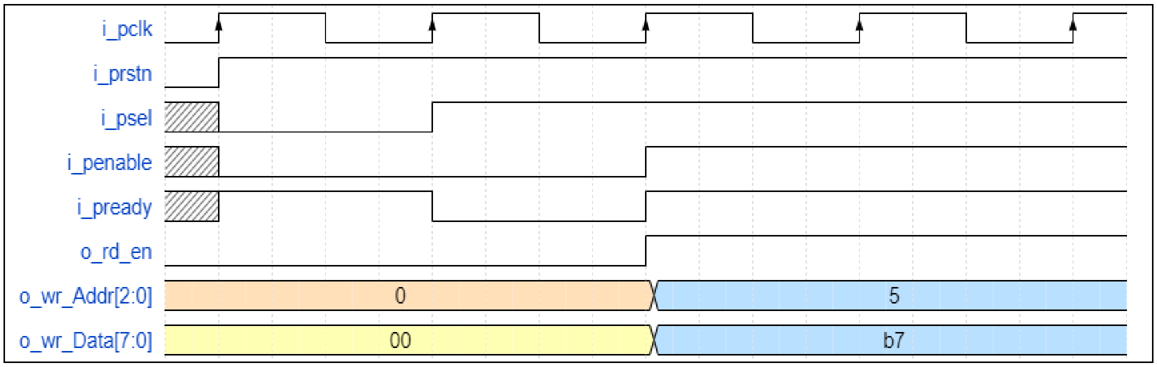

Figure 8.2: Simulation waveform of APB

9. APB with CSR ARCHITECTURE

9.1 Schematic

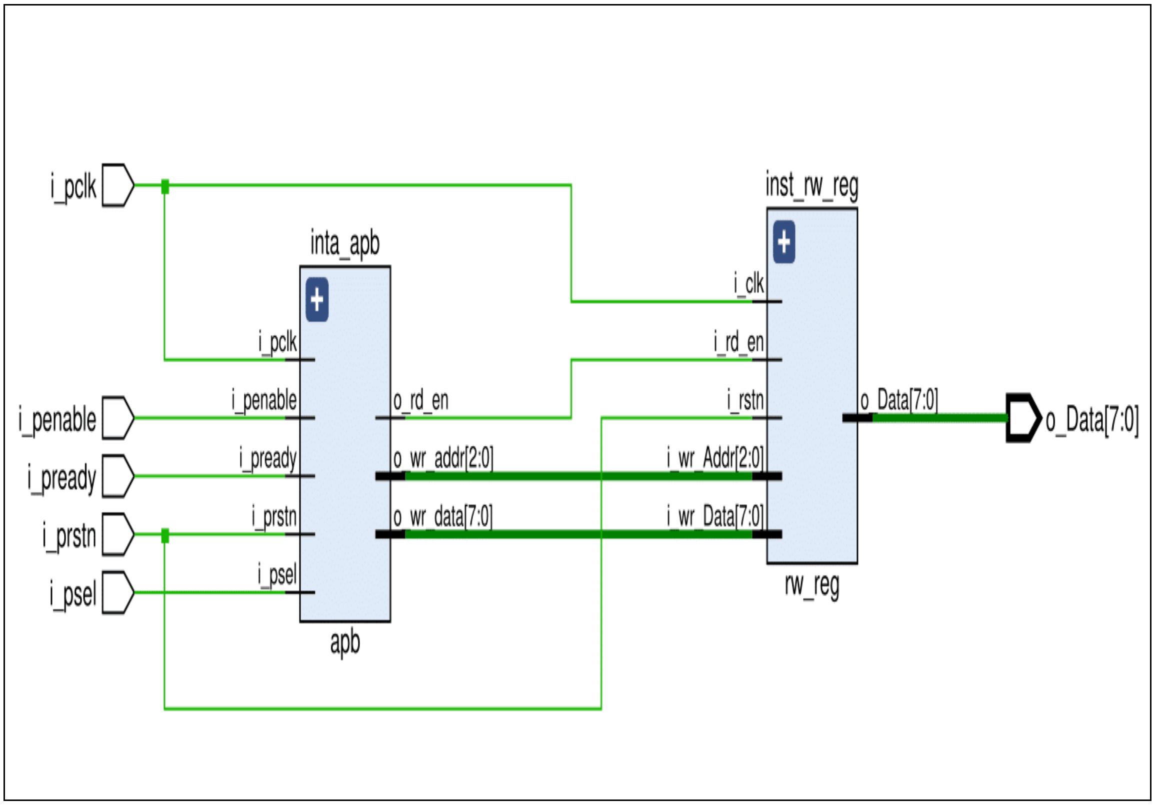

Figure 9.1: APB connected with CSR

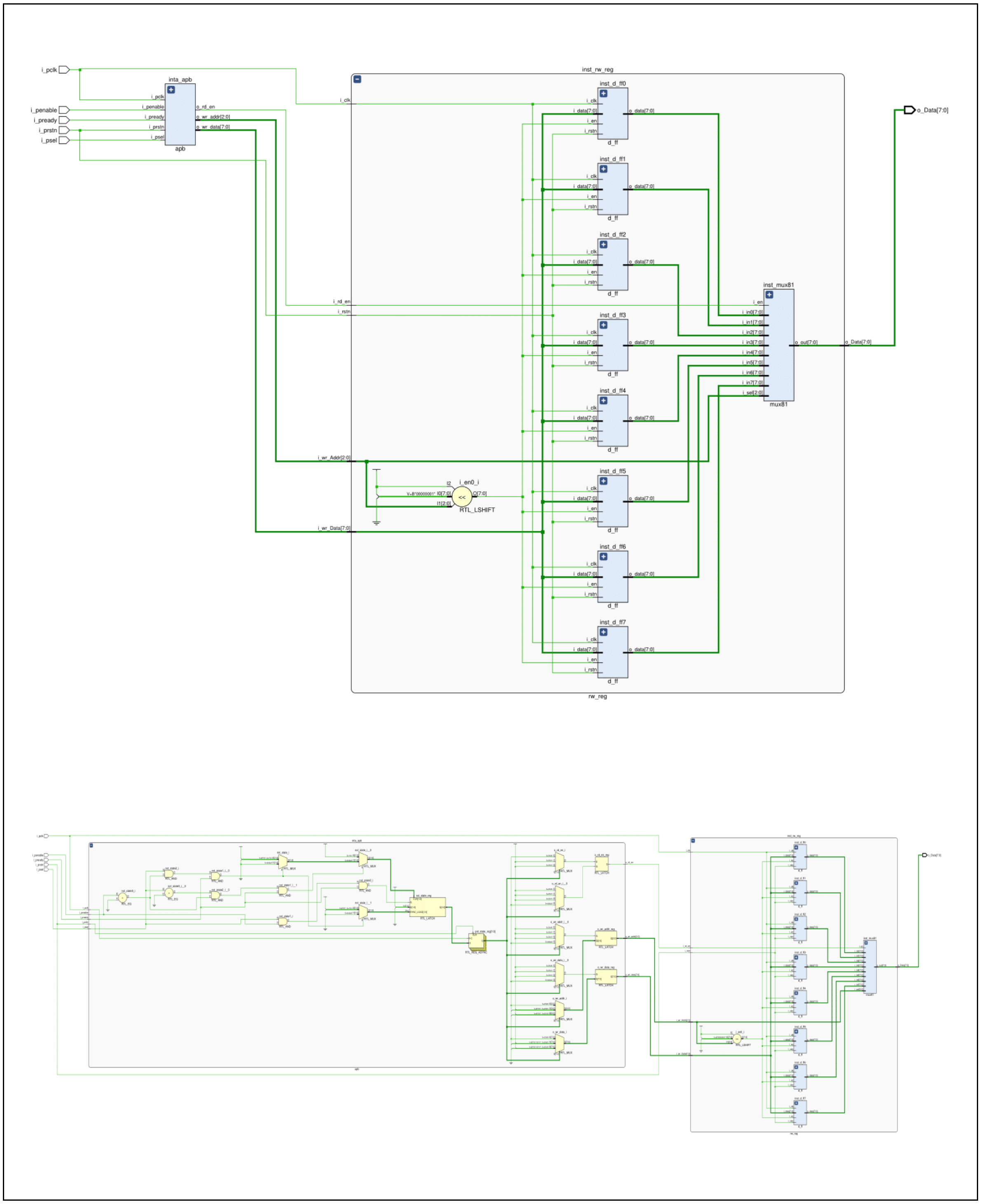

Figure 9.2 : Elaborated Schematic of APB connected with CSR

9.2 Simulation Waveform

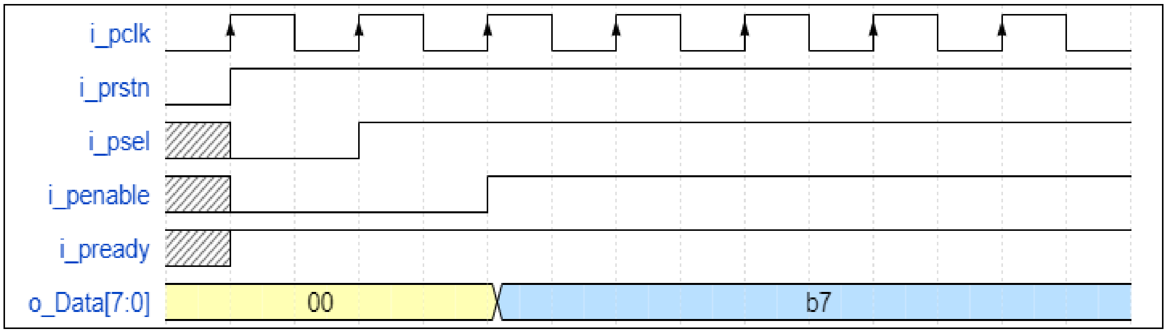

Figure 9.3: Simulation waveform of APB Connected with CSR.

10. APPENDIX

10.1 Source code

```
`timescale 1ns / 1ps
module apb_rw_reg #(
                        parameter ADD_WIDTH  = 3,
                        parameter DATA_WIDTH = 8
                   )

                  (
                   input                                i_pclk,
                       input                            i_prstn,
                       input                            i_pready,
                       input                            i_psel,
                       input                            i_penable,
                       output     [DATA_WIDTH-1 : 0]o_Data
                  );
      wire [DATA_WIDTH-1:0]w_wr_Data;
      wire  [ADD_WIDTH-1:0]w_wr_Addr;
      wire                w_rd_en;

apb inta_apb(
            .i_pclk(i_pclk),
            .i_prstn(i_prstn),
            .i_psel(i_psel),
            .i_penable(i_penable),
            .i_pready(i_pready),
            .o_wr_addr(w_wr_Addr),
            .o_wr_data(w_wr_Data),
            .o_rd_en(w_rd_en)
            );
rw_reg inst_rw_reg(
                   .i_wr_Addr(w_wr_Addr),
                   .i_wr_Data(w_wr_Data),
                   .i_clk(i_pclk),
                   .i_rstn(i_prstn),
                   .i_rd_en(w_rd_en),
                   .o_Data(o_Data)
                   );

endmodule
```

10.2 Simulation code

```
`timescale 1ns / 1ps
module tb_apb_rw_reg();
reg       i_pclk;
reg       i_prstn;
reg       i_psel;
reg       i_penable;
reg       i_pready;
wire [7:0]o_Data;

apb_rw_reg DUT(
                  .i_pclk(i_pclk),
                  .i_prstn(i_prstn),
                  .i_psel(i_psel),
                  .i_penable(i_penable),
                  .i_pready(i_pready),
                  .o_Data(o_Data)
           );

       initial i_pclk  = 1'b0;
       initial i_prstn = 1'b0;
       always #5 i_pclk = ~i_pclk;
       initial begin
```

```
           #5 i_prstn = 1'b1;
              i_psel  = 1'b0;
              i_penable = 1'b0;
              i_pready = 1'b1;



          #10 i_psel  = 1'b1;
              i_penable = 1'b0;



          #10 i_psel  = 1'b1;
              i_penable = 1'b1;

           #50 $finish;

       end


endmodule
```

11. CONCLUSION

- To add Configurability and Observability
- To reduce the Pin count in an IP/SoC.
- Hardware complexity will be reduced.
- Backup plan will be there.

REFERENCES

[1] https://lambdageeks.com/d-flip-flop-circuit-working-truthtable-differences/

[2] https://circuitdigest.com/electronic-circuits/d-flip-flops

[3] https://electrobinary.blogspot.com/2020/06/csr-register-operations-using-apb.html

[4] https://www.ijecce.org/administrator/components/com_jresearch/files/publications/IJECCE_3416_Final.pdf

[5] https://www.researchgate.net/publication/349940087_Design_And_Implementation_Of_Amba_Apb_Protocol

[6] https://web.eecs.umich.edu/~prabal/teaching/eecs373-f12/readings/ARM_AMBA3_APB.pdf

[7] https://www.tutorialspoint.com/what-is-general-register-organization

[8] https://electrobinary.blogspot.com/2020/06/control-and-status-registers-csr.html

[9] https://www.intel.com/content/www/us/en/docs/programmable/691272/21-4-1-0-0/control-and-status-registers.html

[10] https://en.wikipedia.org/wiki/Control/Status_Register

[11] https://www.youtube.com/watch?v=MyS_I7AcDWg

[12] https://youtu.be/3Igv1nSQkE4

[13] https://www.fpga4student.com/2017/02/verilog-code-for-d-flip-flop.html

[14] https://esrd2014.blogspot.com/p/register-file.html

[15] https://www.realdigital.org/doc/ab7587c11f515a7aefd0d72a0eda148d

[16] https://link.springer.com/book/10.1007/978-3-319-56839-3

[17] https://www.researchgate.net/publication/318133814_Decoders_and_Register_Files

[18] https://www.javatpoint.com/verilog-blocking-and-non-blocking

[19] https://www.youtube.com/playlist?list=PLAC_jmBddcjTPEh1UV_ojRJmsx2D9sQXH

[20] https://i.ytimg.com/an_webp/k5zkC7H9AV8/mqdefault_6s.webp?du=3000&sqp=COrOjJwG&rs=AOn4CLDRhtvAGq62hNqokurE2b7Izmq7Aw

[21] http://www.sunburst-design.com/papers/CummingsSNUG2019SV_FSM1.pdf

[22] https://www.youtube.com/watch?v=OVTjSu1jK48

[23] https://www.youtube.com/watch?v=AydxQLwj0fw

[24] https://www.youtube.com/watch?v=kUqEpV0PsmI

[25] https://www.youtube.com/watch?v=ZtM4H8OCWDI

[26] https://www.youtube.com/watch?v=LaZtOScPDQ4

[27] https://github.com/shubhi704/APB-Protocol

[28] https://ieeexplore.ieee.org/abstract/document/4267143

[29] https://www.irjmets.com/uploadedfiles/paper//issue_7_july_2022/28914/final/fin_irjmets1659160932.pdf

[30] http://www.sunburst-design.com/papers/CummingsSNUG2019SV_FSM1.pdf

[31] http://www.sunburst-design.com/papers/CummingsSNUG2000Boston_FSM.pdf

[32] http://www.sunburst-design.com/papers/CummingsSNUG1999SJ_SynthMismatch.pdf

[33] http://www.sunburst-design.com/papers/CummingsICU1997_VerilogCodingEfficiency.pdf

[34] http://www.sunburst-design.com/papers/CummingsSNUG1998SJ_FSM.pdf

[35] http://www.sunburst-design.com/papers/CummingsHDLCON2001_Verilog2001.pdf

[36] https://www.researchgate.net/profile/Neeraj-Shukla-6/publication/269669106_Design_and_Verification_of_AMBA_APB_Protocol/links/603cd29c299bf1cc26fc397b/Design-and-Verification-of-AMBA-APB-Protocol.pdf