**ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH**

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN – ĐIỆN TỬ

**BỘ MÔN ĐIỆN TỬ**

---------------o0o---------------

****

**CAPSTONE PROJECT 2 REPORT**

**DESIGN AMBA AHB -APB SPECIFICATION**

**COMMUNICATE IN UART**

|  |  |
| --- | --- |
| **Instructor:** | **Dr. Trần Hoàng Linh** |
| **Student’s name:** | **Võ Viết Hưng** |
| **Student’s ID:** | **2051076** |

**Hồ Chí Minh city, July 31th, 2024**

**TABLE CONTENT**

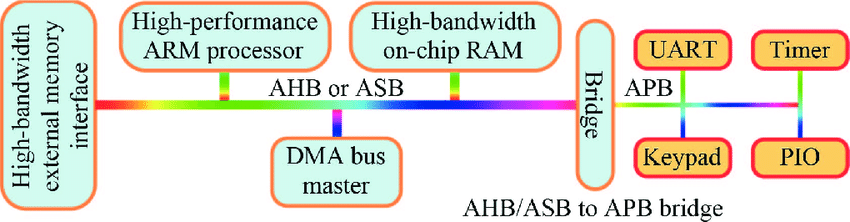
[1. **INTRODUCTION** 1](#_Toc501486126)

[2. **PROGRESS OF PROJECT** 5](#_TOPIC’S_CONTENT)

[3. **REFERENCES** 13](#_4._REFERENCES)

# 1. INTRODUCTION

## *1.1. Topic’s selection reason*

In this Capstone Project 2, I chose to design an AMBA (Advanced Microcontroller Bus Architecture) system featuring AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus) interfacing with UART (Universal Asynchronous Receiver-Transmitter) stems from the growing need for efficient and scalable communication in embedded systems.

***Figure 1.1: A typical AMBA based SoC design***

To more specific, AHB provides high-speed communication suitable for processors and memory interfaces, while APB offers a simpler, lower-power connection to peripherals. By integrating UART, a widely used communication protocol, into this setup, the project aims to facilitate robust and reliable serial communication between the system and external devices. This design ensures seamless data transfer across different bus protocols, enhancing the overall performance and versatility of the SoC (System on Chip). The project will focus on creating a cohesive and efficient bridge between these components, making it a critical study for modern embedded system applications.

## *1.2. Topic’s aim*

The primary objective of this project is to design an efficient AMBA (Advanced Microcontroller Bus Architecture) system that integrates AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus) to facilitate communication with UART (Universal Asynchronous Receiver-Transmitter). Specifically, the project aims to develop a robust AHB-APB bridge that ensures seamless data transfer between high-speed processors and low-power peripheral devices. Additionally, the design will focus on achieving reliable UART communication for serial data exchange with external devices, enhancing system versatility. By meeting these objectives, the project aspires to contribute to the development of scalable and efficient embedded systems, ultimately improving overall performance and functionality within System on Chip (SoC) architectures.

# 2. CONTENT OF TOPIC

The project focuses on designing an AMBA (Advanced Microcontroller Bus Architecture) system that incorporates AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus) for effective communication with UART (Universal Asynchronous Receiver-Transmitter). Understanding the operating principle of this design is essential for appreciating how these components work together to enhance system performance in embedded applications.

At the core of this architecture, the AHB serves as the primary high-speed bus that facilitates rapid data transfer between the central processing unit (CPU) and memory modules. This allows the system to execute complex computations and handle large volumes of data efficiently. The AHB is designed to support high bandwidth and low latency, making it suitable for performance-critical operations.

In contrast, the APB is optimized for connecting low-power peripherals, such as sensors and control devices. This bus is essential for maintaining energy efficiency, as it operates at lower speeds and consumes less power compared to the AHB. By using the APB for peripheral communication, the overall power consumption of the system is significantly reduced, which is crucial for battery-operated or energy-sensitive applications.

| **Feature** | **AHB (Advanced High-performance Bus)** | **APB (Advanced Peripheral Bus)** |
| --- | --- | --- |
| **Purpose** | High-speed communication for processor and memory | Low-power, low-speed peripheral communication |
| **Bandwidth** | High bandwidth | Low bandwidth |
| **Latency** | Low latency | Higher latency compared to AHB |
| **Complexity** | More complex | Simpler design |
| **Power Consumption** | Higher | Lower |
| **Usage** | Main bus for CPU, memory, and high-speed devices | Connecting peripheral devices like sensors |
| **Clocking** | Synchronous with the system clock | Can be asynchronous or synchronous |
| **Data Transfer** | Supports burst transfers and pipelined operations | Simple, single data transfers |
| **Control Signals** | More control signals (e.g., HTRANS, HBURST, HPROT) | Fewer control signals |
| **Protocol Complexity** | More complex protocol | Simpler protocol |
| **Implementation Cost** | Higher | Lower |
| **Example Devices** | CPUs, DMA controllers, high-speed peripherals | UART, GPIO, timers, lower-speed peripherals |

***Table 2.1: A typical AMBA AHB and APB short comparison***

To bridge the gap between these two buses, the AHB-APB bridge is implemented. This component ensures seamless data transfer between the high-speed AHB and the low-power APB, allowing the system to manage different data rates and operational requirements effectively. The bridge translates signals and controls the flow of data, enabling the CPU to communicate with peripheral devices without any bottlenecks.

Furthermore, UART communication plays a vital role in this design by providing a reliable method for serial data exchange with external devices. UART is widely used due to its simplicity and effectiveness in applications such as telemetry and device control. By integrating UART into the AMBA architecture, the system can easily interact with a variety of external components, enhancing its versatility and functionality.

In summary, the operating principle of the AMBA system design with AHB-APB interfacing and UART communication is centered around optimizing data transfer and power efficiency. The combination of a high-speed bus for critical operations and a low-power bus for peripherals, along with a robust bridging mechanism and reliable serial communication, creates a cohesive system that meets the demands of modern embedded applications.

# 3. PROGRESS OF PROJECT

## *3.1. AHB – APB Bridge*

### ***3.1.1. FSM Diagram***

***Figure 3.1.1:*** *State machine for AHB to APB interface*

Description:

* + - * ST\_IDLE stage: During this state the APB buses and PWRITE are driven with the last values they had, and PSEL and PENABLE lines are driven LOW.
* *The ST\_IDLE state is entered from:*

*+) reset, when the system is initialized*

*+) ST\_RENABLE, ST\_WENABLE, or ST\_IDLE, when there are no peripheral transfers to perform.*

* *The next state is:*

*+) ST\_READ, for a read transfer, when the AHB contains a valid APB read transfer.*

*+) ST\_WWAIT, for a write transfer, when the AHB contains a valid APB write transfer.*

* ST\_READ stage: During this state the address is decoded and driven onto PADDR, the relevant PSEL line is driven HIGH, and PWRITE is driven LOW. A wait state is always inserted to ensure that the data phase of the current AHB transfer does not complete until the APB read data has been driven onto HRDATA.
* *The ST\_READ state is entered from ST\_IDLE, ST\_RENABLE, ST\_WENABLE, or ST\_WENABLEP during a valid read transfer.*
* *The next state will always be ST\_RENABLE.*
* ST\_WWAIT stage: This state is needed due to the pipelined structure of AHB transfers, to allow the AHB side of the write transfer to complete so that the write data becomes available on HWDATA. The APB write transfer is then started in the next clock cycle.
* *The ST\_WWAIT state is entered from ST\_IDLE, ST\_RENABLE, or ST\_WENABLE, during a valid write transfer.*
* *The next state will always be ST\_WRITE.*
* ST\_WRITE stage: During this state the address is decoded and driven onto PADDR, the relevant PSEL line is driven HIGH, and PWRITE is driven HIGH. A wait state is not inserted, as a single write transfer can complete without affecting the AHB.
* *The ST\_WRITE state is entered from:*

*+) ST\_WWAIT, when there are no further peripheral transfers to perform.*

*+) ST\_WENABLEP, when the currently pending peripheral transfer is a write, and there are no further transfers to perform.*

* *The next state is:*

*+) ST\_WENABLE, when there are no further peripheral transfers to perform.*

*+) ST\_WENABLEP, when there is one further peripheral write transfer to perform.*

* ST\_WRITEP stage: During this state the address is decoded and driven onto PADDR, the relevant PSEL line is driven HIGH, and PWRITE is driven HIGH. A wait state is always inserted, as there must only ever be one pending transfer between the currently performed APB transfer and the currently driven AHB transfer. See the write transfer timing diagrams in the AMBA Specification (Rev 2.0) for more details.
* *The ST\_WRITEP state is entered from:*

*+) ST\_WWAIT, when there is a further peripheral transfer to perform.*

*+) ST\_WENABLEP, when the currently pending peripheral transfer is a write, and there is a further transfer to perform.*

* *The next state will always be ST\_WENABLEP.*
* ST\_RENABLE stage: During this state the PENABLE output is driven HIGH, enabling the current APB transfer. All other APB outputs remain the same as the previous cycle.
* *The ST\_RENABLE state is always entered from ST\_READ.*
* *The next state is:*

*+) ST\_READ, when there is a further peripheral read transfer to perform.*

*+) ST\_WWAIT, when there is a further peripheral write transfer to perform.*

*+) ST\_IDLE, when there are no further peripheral transfers to perform.*

* ST\_WENABLE stage: During this state the PENABLE output is driven HIGH, enabling the current APB transfer. All other APB outputs remain the same as the previous cycle.

*The ST\_WENABLE state is always entered from ST\_WRITE.*

*The next state is:*

*+) ST\_READ, when there is a further peripheral read transfer to perform.*

*+) ST\_WWAIT, when there is a further peripheral write transfer to perform.*

*+) ST\_IDLE, when there are no further peripheral transfers to perform.*

* ST\_WENABLEP stage: A wait state is inserted if the pending transfer is a read because, when a read follows a write, an extra wait state must be inserted to allow the write transfer to complete on the APB before the read is started.
* *The ST\_WENABLEP state is entered from:*

*+) ST\_WRITE, when the currently driven AHB transfer is a peripheral transfer.*

*+) ST\_WRITEP, when there is a pending peripheral transfer following the current write.*

* *The next state is:*

*+) ST\_READ, when the pending transfer is a read.*

*+) ST\_WRITE, when the pending transfer is a write, and there are no further transfers to perform.*

*+) ST\_WRITEP, when the pending transfer is a write, and there is a further transfer to perform.*

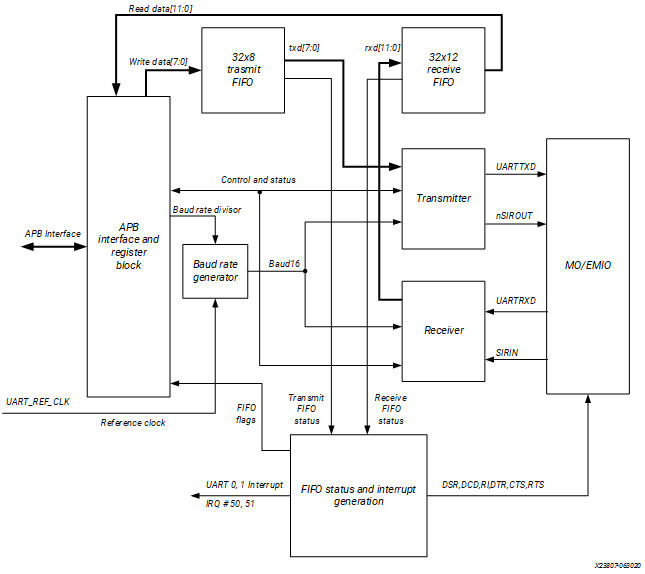
### ***3.1.2. Block Implementation***

***Figure 3.1.2:*** *Block diagram of AHB to APB Bridge (AHB Slave)*

## 3.2. UART APB Implementation

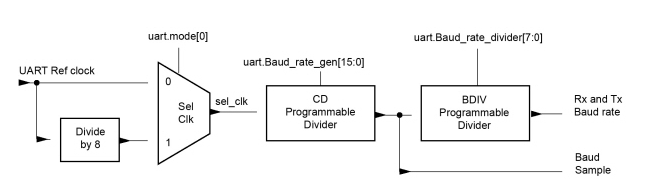
**Figure 3.2.1: General block for pin out & in signal of APB UART**

As I have researched during this week, inside APB UART (APB Slave), there are several sub-blocks, took different responsibility such as (based on below figure):



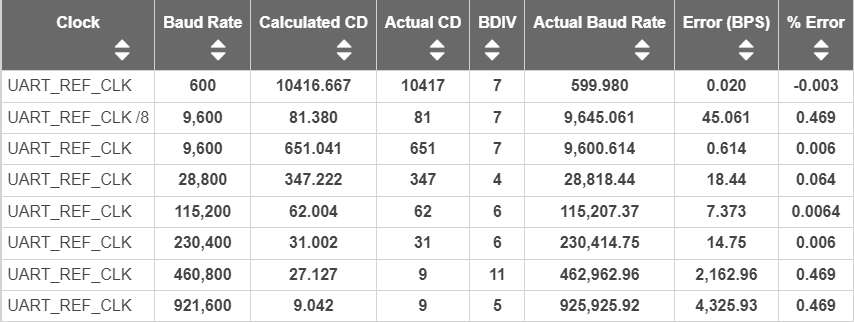
***Figure 3.2.2: Detail Block Diagram of APB UART***

* **Baudrate Generator** (theory): furnishes the bit period clock, or baud rate clock, for both the receiver and the transmitter. The baud rate clock is implemented by distributing the base clock UART\_REF\_CLK and a single cycle clock enable to achieve the effect of clocking at the appropriate frequency division.

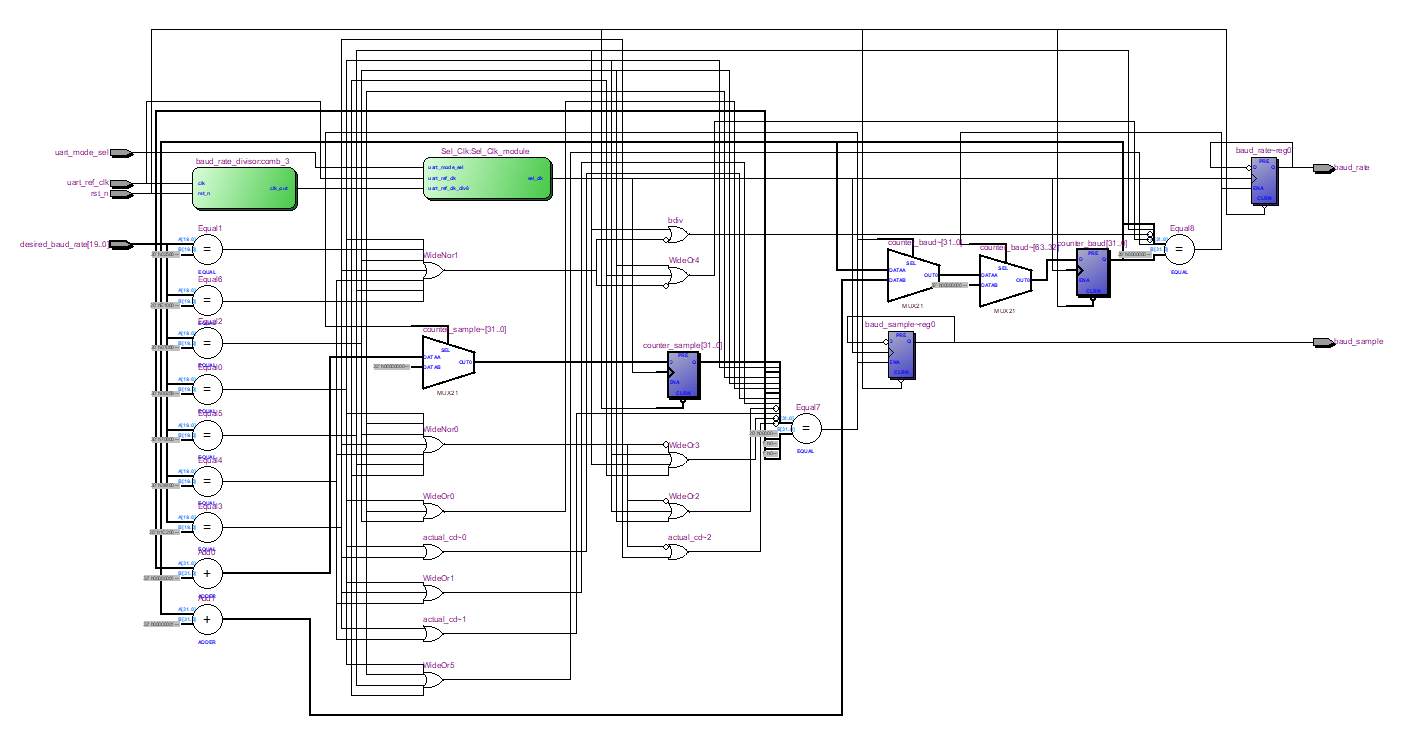
 ***Figure 3.2.3: Block flow design (compenent inside Baud Rate Generator)***

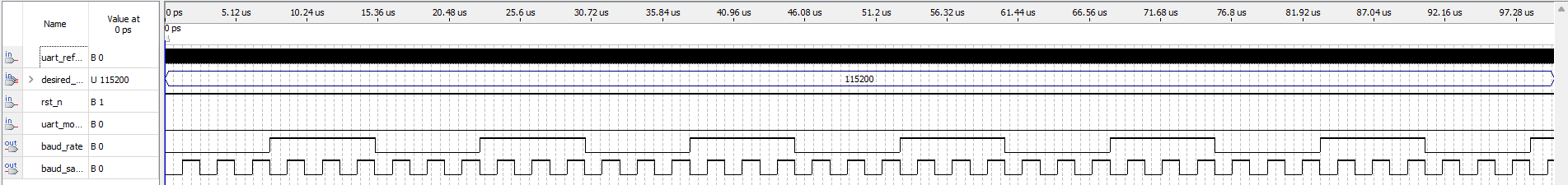
Baud Rate Divisor = UARTx\_REF\_CLK/8= [DIVINT] . [DIVFRAC] (\*\*)

**Note**: This is the value come out from Divide\_8 block due to Figure 1.2

During implementation, CD value component of Baudrate Generator block is not applied toq floating point yet, so there would be error of displaying baud rate exactly value. I already has the way to solve this problem based on the (\*\*) function, in the next few weeks, after I finished with the UART APB block, I’ll comeback and solve this error different problems.

***Table 3.2: UART Parameter Value Examples***

* Come to General Block diagram of BAUD RATE GENERATOR: Here is the block structure that I have based on <https://docs.amd.com/r/en-US/ug1085-zynq-ultrascale-trm/Baud-Rate-Generator>
* ***Figure 3.2.4: block diagram of UART Baudrate Generator***

And the following ModelSim simulation for making sure that this block run correctly.

***Figure 3.2.5: Simulation of UART baud rate generator***

**Where**: In this example, I used UART Baudrate Generator to create an baudrate at 115200 (baud/s), so as result, I insert the value of Baudrate into desire\_baud\_rate (signal) anh this block will do its responsibility. For more details, as I inputted 115200 into desire\_baud\_rate signal, this block generated **CD** (clock divider) and **BDIV** (baudrate divider) so that by dividing the uart\_ref\_clk by 62 time, meaning each 62 clocks counting, the block will inverse the status of previous **clk\_div** value and generate **baud\_sample** and for each 6 times of clk\_div changing state, the baud\_rate output signal will inverse its state. And that is the general operation for this UART Baudrate Generator.

# 4. IMPLEMENTATION PLAN

* ***Dissertation time***: 2 months
* ***Detail Proces Plan:***

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Content** | **July** | **August** |
| 1 | ***Explore possible designs*** | ✓ |  |
| 2 | ***Design implementation*** | ✓ |  |
| 3 | ***Design implementation*** |  |  |
| 4 | ***Design implementation*** |  |  |
| 5 | ***Complete design*** |  |  |
| 6 | ***Insert Application check*** |  |  |
| 7 | ***Check the design*** |  |  |
| 8 | ***Report*** |  |  |

# 5. REFERENCES

1. ***AMBA AHB Protocol Specification*** <https://developer.arm.com/documentation/ihi0033/latest/>
2. ***AHB Example AMBA System Technical Reference Manual*** <https://developer.arm.com/documentation/ddi0170/a/I967114>
3. ***DirectCore Advanced Microcontroller Bus Architecture - Bus Functional Model*** <https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/UserGuides/CoreAMBA_BFM_UG.pdf>
4. ***Effective Design and Implementation of AMBA AHB Bus Protocol using Verilog*** <https://www.researchgate.net/publication/337510558_Effective_Design_and_Implementation_of_AMBA_AHB_Bus_Protocol_using_Verilog>
5. ***AMBA AHB Protocol*** <https://fr.scribd.com/doc/41197279/AMBA-AHB-Protocol-Presentation>
6. ***PrimeCell UART (PL011) Technical Reference Manual*** <https://developer.arm.com/documentation/ddi0183/g>
7. ***Design and FPGA Implementation of UART Using Microprogrammed Controller***

<https://www.researchgate.net/publication/282030059_Design_and_FPGA_Implementation_of_UART_Using_Microprogrammed_Controller>