Digital Design Group report

Group Number: 7

Group members:

|  |  |
| --- | --- |
| * Luke Zhang (az16408） | * Violet Yuan (by17205) |
| * Jack Zhao (fz17963) | * Yushi Wang (yw17822) |
| * Juncheng Shang (js16481) | * Muzixiang Xiao (mx17936) |

Content

1. Introduction
2. Group composition and task division
3. Project plan
4. Architecture of the Data Processing module

* Block level sketch of the main logic components
* State Diagram of FSM

1. Architecture of the Command Processing module

* Block level sketch of the main logic components
* State Diagram of FSM

1. Introduction

In this peak detector system, one of the important parts is the data processor which takes the part of requesting and processing the data from the data processor and send the detected or required data results to the command processor in order to be presented by the PC monitor which is connected to a FPGA with UART and communicate through the UART. The propose of the data processor briefly is to receive the data, monitor the given data and find the peak value along with three values before and after it from the data flow respectively, and send the peak value and the index of it to the command processor in order to do the next process until the sequence is done with 500 values which is indicated.

The entire block of data processor is connected with the data generator and command processor. Besides the basic inputs of clock and reset in every part of the components as a synchronize system, data processor receives 8-byte data by the in and out control signal from the data generator, and then transmits the detected results.

1. Group composition and task division

There are 6 members in this group. Given the data generator, receiver and transmitter module, the task for this Assignment was divided into two parts according to the block level architecture of the peak detector module, namely Data Processor and command processor. As a result, it has been discussed and decided that the task will be carried out within two smaller group; the team members who are responsible for designing the Data processor are Violet Yuan, Bill Wang, Leon Xiao, and the members who are responsible for implementing the Command processor part are Luke Zhang, Jack Zhao, Zed Shang. The detailed group task division can be seen as follows:

**Data Processor Sub-Team:**

* *Work requirements*

In case of connecting the command processor, the start signal is a enable of the whole data processor which is from command processor with numWords which indicates how many data should be processed. The binary peak value and its index in BCD form are transmitted to the command processor by dataReady and byte signal when peak and 3 bytes ahead and behind are formed. Meanwhile, 12-bit maxIndex, dataResults, and seqDone will transmit to command processor as well. To achieve the goals required, three main functional blocks are needed. First of all, comparing the peaks and recording the 3 bytes before and after each peak. Register7 is used to compare the peaks and record them and 3 bytes after the peak. Register4 is used to record 3 bytes before current byte at any time. Secondly, counting the number of bytes needed to process and the number at which peak occurs. A BCD counter is used to record the number needed as well as counting up. Register12 is used to record the peak index. A T\_R block is used to request data by using hand shake protocol. Also, a controller block is made to control the above 5 functional blocks.

There are three register blocks using (register4, register7, and register12) to satisfy the requirements of a peak detector on the side of the data process. The number 4, 7, and 12 indicates that the register is associated with 4 bytes, 7 bytes, and 12 bits from the data flow and the index. As a synchronised system in this particular mechanism, all of the data transition should take place only when there is a clock falling edge in order to keep its consistency.

* *Peak detection and storage*

In aspect of the actual data selecting, there are in total of 7 bytes to be detected (including 3 bytes before and 3 bytes after the peak). If the data flow appears a value that is larger than the peak whose function is achieved by a comparator inside of register 7, when the next clock falling edge, the peak will be replaced by data and the 3 value before will be replaced by the value in register4. Meanwhile, the 3 bytes after the peak will also be overwritten one by one at each falling edge by a local 2 bit counter. The whole series of 7 bytes data results is formed after 3 clock cycle from peak byte.

REGISTER4 takes in charge of the three bytes before the current byte. Every time the new data is in, it keeps recording simultaneously. When the previous peak value is outnumbered by the current data value and becomes the current peak following the clock falling edge, the 3 bytes ahead of the current peak will be overwritten and be regarded as the new 3 bytes ahead of the current peak and transmitted to register 7.

* *Counting*

A BCD counter is required to serve two main functions in the Data Processor. First, as a counter, this component is used to take count of the events, which is the number of the data that appears from the data source, also known as the index of this number. Secondly, keep in track of the number of data coming in and knowing which number the input is at, hence the order of the number is also a function for this counter using the index. As the result of the input and output are both BCD, this counter is designed to be taking BCD as the counting mode. The counter increments by 1 in the first digit of the index each time a new data is in and when the first digit has counted to 9, the first digit changes back to 0 and the second digit of the index increments by 1 and so does the third digit with the same process when the second digit is counted to 9 whereas each digit is represented by the binary coded decimal which is in total of 12 bits with each number using 4 bits.

Register12 is connected and serves the BCD counter. After the reset has cleared the counter and enable is on, the index will change every time when the clock switch from high to low while the data is flowing in.

* *Communication with data generator*

Within the block of Data Processor, T\_R is used to convert logic level into hand shake protocol and vice versa. When T is produced by the controller, T\_R will set ctrlOut to its opposite value by a T flip-flop. T\_R will also record the ctrlIn value at each falling edge. Output R will be asserted if the received ctrlIn is different from the recorded one until the next falling edge. The recorded values and all the outputs will be set to 0 if reset is asserted.

* *Control*

The controller inside the Data Processing unit is a state machine used to organise the peak detection process. Four states are made to achieve this. The state transition is on the state diagram and will only be mentioned briefly. Two More outputs T and seqDone are involved in the state machine which remains low in each state if not mentioned in the following paper. INIT state is an idle state which will transit into FIRST state when start is high. FIRST state enables the peak detection. Output T will be high to which forces T\_R to require data from data generator. After every assertion of start, the state machine will be in SECOND state which is idle until the next start. Furthermore, the state machine will be set to INIT state when reset is asserted. After processing all the bits, the state machine will be in THIRD state. Output seqDone will be high to inform the command processor for one clock cycle.

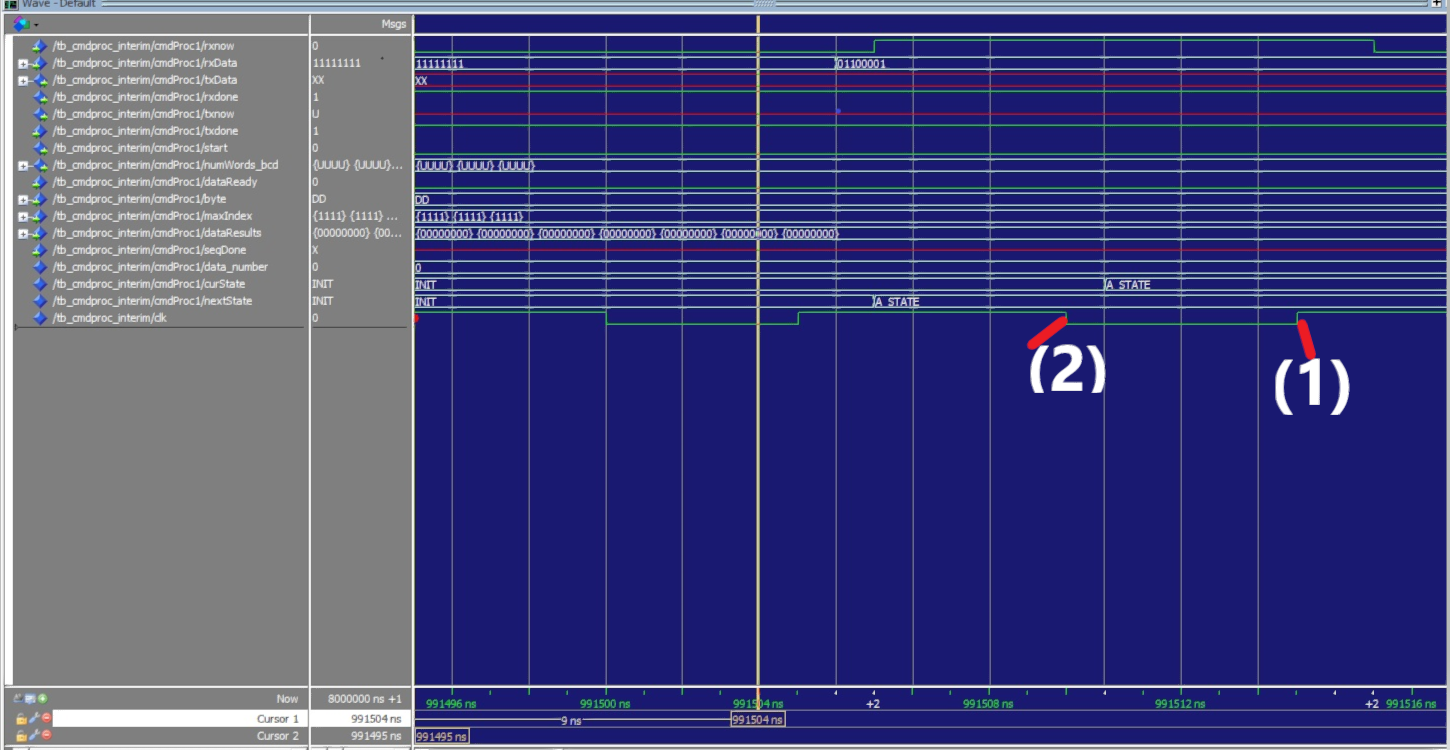
**It is noteworthy that *dataConsume.vhd* is used to link all the blocks above.**

* Work Allocation:

Violet Yuan is in charge of register12, T\_R, report writing, code debugging. Muzixiang Xiao is responsible for state diagram and block diagram plotting, controller, grammarian. Yushi Wang conducts overall plan and structure, register7, register4, BCDcounter, report layout.

**Command Processor Sub-Team:**

* Jack Zhao was responsible for developing and troubleshooting, including setting the all the ports and initializing the INIT state then built the links to the next several states (A\_STATE, L\_STATE, P\_STATE) based on different data received from the receiver. Then found the different binary bits numbers matched to the certain hex numbers for “ANNN”, which corresponds to states A\_STATE, N\_STATE1, N\_STATE2. Meanwhile, BCD adopt makes “ANNN” convert into corresponding numbers in each state.

**Figure 1: Simulation wave diagram of the Command processor**

At the beginning of each state (before the transition of states), rxDone is set to 0 and only goes back to 1 until the current state transition is finished, which means the data transferred has not been read unless all bits of the current byte from receiver are read.

In addition, due to the settings of the TestBench, data are only read after the period of rxNow turns from 0 into 1 (991506 ns), based on the observation of clock event, if the clock is set as the rising edge, rxNow which is after (1) cannot be read, (as shown in the **Figure 1**)the logic block will be crashed if there are any delays incurs. However, for the falling edge of clock, information which is after (2) could be read, there is a half-cycle delays allowed for the mechanism, and normally, the delay is possibly within this value, which can make the mechanism run safely.

* Zed Shang did some states for the transmission of data between data processor and command processor. Firstly, set the start as 0 initially, which means that the process of transmission is not started, when the state is transferred to Trans\_State, as the output “start” turns to 1, the transmission of data gets started, to avoid the data copied consistently at the transmitter when the data is transmitting, txNow must be reset to 0 to cut the connections between the command processor and transmitter. Once the command processor receives a signal that current byte is done (dataReady is 1), transmitter copies the data in the command processor when a send operation is triggered (send is high). Then “ready” is called and “send” operation stops, which means txDone should be set to high, and the value of variable ‘I’ should be checked to compare with the data(Data\_Number) received previously while the clock is still working, if ‘I’ is smaller than this data, which means the data transmitted is incomplete, then the machine goes back to Trans\_state and increase ‘I’ by 1 until ‘I’ is equal to this data(complete data), once this condition is satisfied, which means the commands is implemented successfully and the processor should go back the initial state, certainly, the variable ‘I’ requires to be reset as 0 as well. He also involved in debugging and testing for the main Command processor modules. Finally, after all these states finished, a Trans\_STATE required is to make sure that there is a time gap existed until the data processor finished one-byte processing.
* Luke Zhang was part of the developing and designing process, he designed and sorted out the output of the peak value(P\_STATE) as well as the peak sequence(L\_STATE). To get the value of the peak and its maximum index, the dataResults from txData need to be obtained at first, then print txData, followed by the value of the maximum index value after a tab. As a result, the expected output format should be txData + Tab + maxIndex. After that, it will go back to the initial state immediately.

As for the L\_STATE, txNow need to be set to ‘0’ initially, upon the detection of the condition seqDone=’1’, one byte of data will be transmitted and txNow will be set to ‘1’. In state L\_w8, the condition txDone=’1’ and the falling edge of the clock need to be satisfied in order to enter this state, and it will switch between L\_STATE and L\_w8 repeatedly as long as I<1 is true since each state switch transmission means a new byte coming through, until it reaches the condition I=7, which means the end of transmission of full 7 byte if the data. After that, it will go back to the initial state and ready to transmit a new sequence of data according to the requirement of the command line that received from the transmitter.

1. Project Plan

Approximately one-hour group meeting every week. Exchange ideas and debate between team members. Discuss what we have done so far and make plans for next week, also sort out any confusions or problems that may occur. The detailed work progress and between group members can be seen in **Table 1**, **Table 2** and Gantt chart (**Figure 2**).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **TASK NAME** | **START DATE** | **END DATE** | **DURATION (WORK DAYS)** | **TEAM MEMBER** | **PERCENT COMPLETE** |
|
| **interim submission** | |  | | | | |
|  | Command processor | 20/02/2019 | 25/03/2019 | 33 | Luke | 100% |
|  | Command processor | 20/02/2019 | 24/03/2019 | 32 | Jack | 100% |
|  | Command processor | 20/02/2019 | 23/03/2019 | 31 | Zed | 100% |
|  | Data processor | 20/02/2019 | 25/03/2019 | 33 | Violet | 100% |
|  | Data processor | 20/02/2019 | 27/03/2019 | 35 | Leon | 100% |
|  | Data processor | 20/02/2019 | 24/03/2019 | 32 | Bill | 100% |

**Table 1: General working progress of individuals**

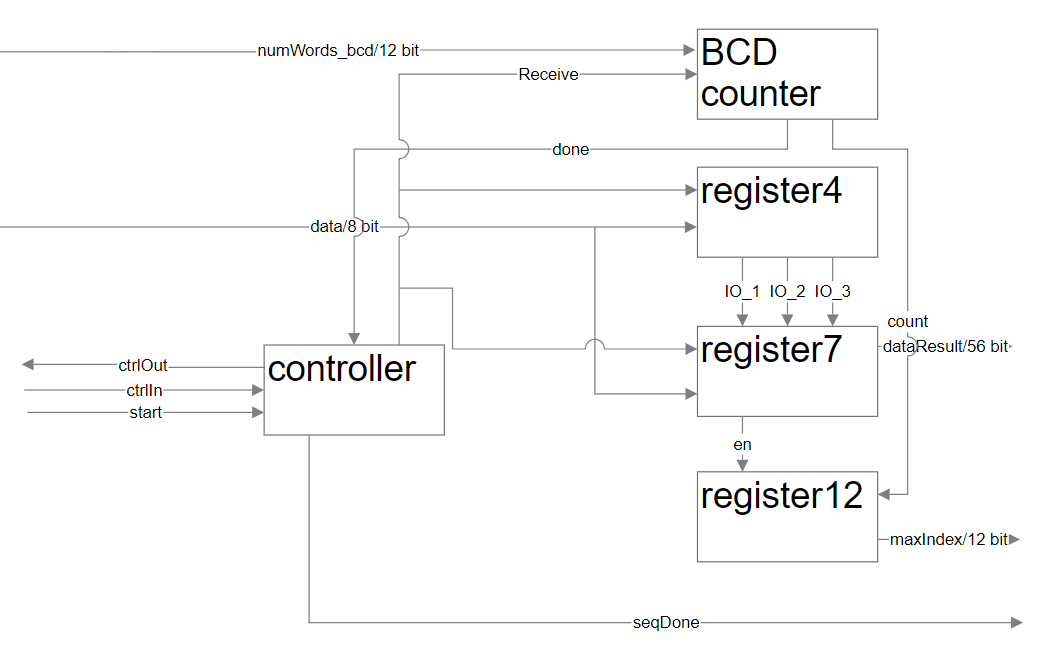
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **M** | **T** | **W** | **Th** | **F** | **Milestones** |
| WEEK 16 |  |  | ✔ |  |  | Task divided |
| WEEK 17 |  |  | ✔ |  |  | Implementations began |
| WEEK 18 |  |  | ✔ |  |  | State transitions finished |
| WEEK 19 | ✔ |  |  |  |  | Logic & Simulations cleared |
| WEEK 20 |  |  |  |  | ✔ | Implementation Progress updated |
| WEEK 21 | ✔ |  |  |  |  | Final discussion of signal 'start' |

**Table 2: Weekly group meeting schedule**

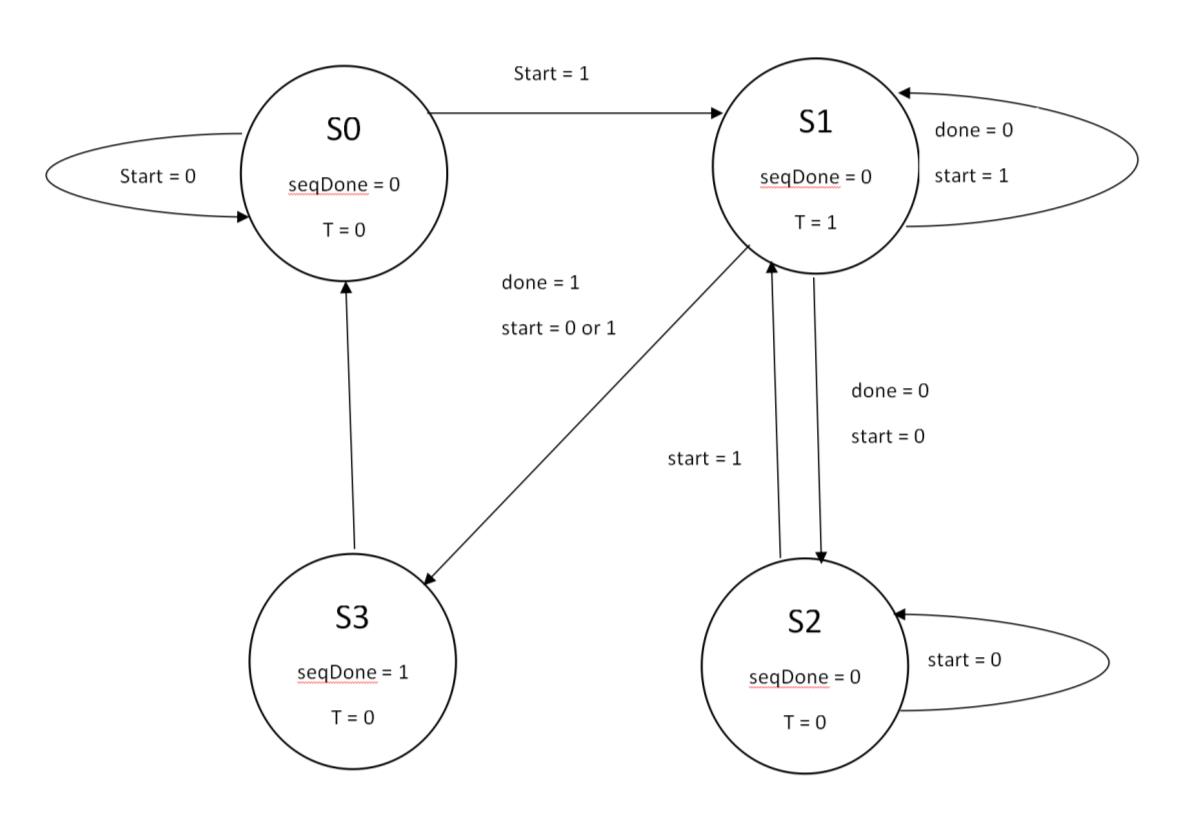
**Figure 2: Gantt chart of the team work progress**

1. Architecture of the Data Processing module

* Block level sketch of the main logic components

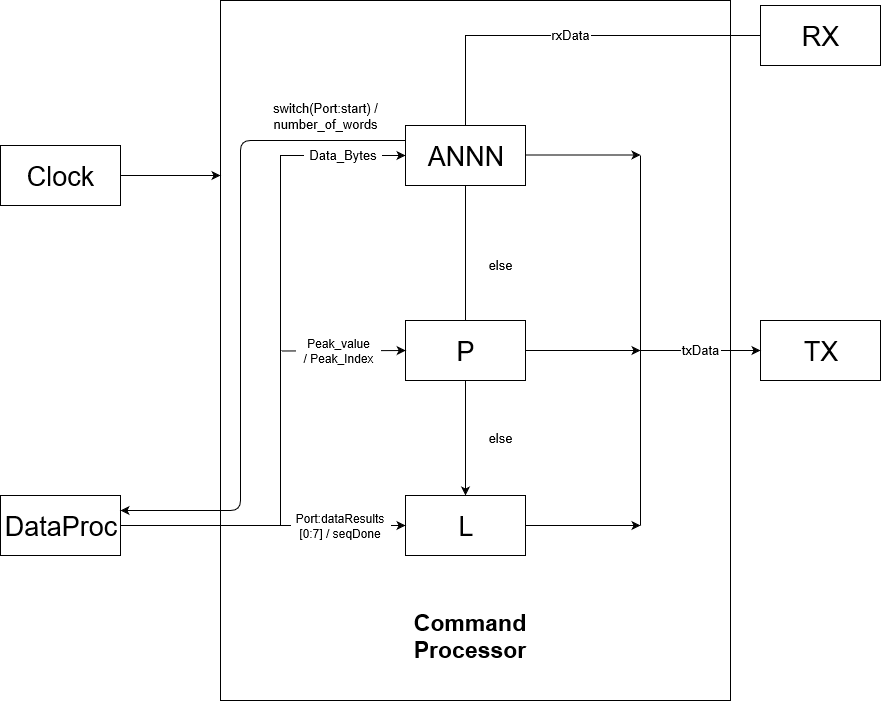


* State Diagram of FSM



1. Architecture of the Command Processing module

* Block level sketch of the main logic components



* State Diagram of FSM

