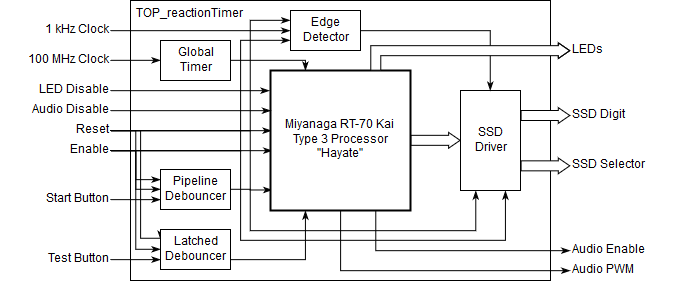
# Features

* Implement the four period of the Reaction Timer specification on Nexys4 DDR
* [Advanced] Shows the best record time and LED animation at the idle state
* [Advanced] Automatically return to idle state after display the reaction time for 10 seconds
* [Advanced] Display the reaction time as a 7-digit decimal floating point number in second
* [Advanced] Using Linear Congruential Generator as the random number generator to set random event time
* [Advanced] The recent reaction time would be to flash if it is the best time
* [Advanced] If tester pressed the button before the reaction signal, the reaction timer will mark this as a failure.
* [Advanced] Support mono audio output as the optional reaction signal

# Architecture

The following diagram shows the top level design of the reaction module.



The input wires of the global module are 4 switches, 2 buttons and system 100MHz clock. One switch uses as reset, one uses as enable, one uses as audio output disable, and the other one uses as LED output disable. One button uses as the test start button and the other one uses as test button. The output wires are 16 LEDs, 7-segement display and mono audio outputs.

The top module architecture is designed for processing the input and output signal directly to the FPGA board. All the input and output pins are chosen to be used CMOS 3.3V standard. The clock used in this design is the 100MHz clock provided by the crystal on the board. An 1kHz clock generated from the 100MHz clock is used as the clock for the Seven-Segments Display to select the digit.

The edge detector has the same logic as the one we used in Lab 4.

## Modules at Top Level

### Input Signal Debouncing

For all the switches used in this reaction timer, it doesn’t need any debouncer. But for those two buttons, they need different debouncing methods.

For the start button, it is only used as a signal for starting the test. So the debouncer for start button is just the sample pipeline debouncer as we implemented in the lab. The pipeline is described as a single register. The different from the lab version is the pipeline level could be modified by a parameter. If we still use the way as what we did in the Lab, the value used to detect the button pressed would be 2n+1-1 in decimal. To reduce the work of FPGA, we store the inverse of the button instead of its original value. When the button pressed for a while, the value of the pipeline should be 0. The debouncer module also uses logic equal to avoid the high impedance value comparison. Due to the limitation of the integer is 32-bit as default, hence this module could support up to 32-level pipeline. In the implementation of reaction timer, this level has been set to 5.

For the test button, it needs for lower latency to get accurate result. Here, we use a latched debouncer for the test button, which allows us to reduce latency within 10ns. The edge detector would add another 10ns delay to the signal. Hence, the total latency of the test button pressed is 20ns for these two modules. However, this won’t affect the precise of our final result. It will explain further later.

### Global Timer

The global timer module is a simple counter start running at the very beginning. It works like the UNIX timestamp. There is a 32-bit unsigned register and initializes as 0. This register increase 1 for each 100MHz rising edge. Hence, it would warp around every 42,949,672,950 nanoseconds, which is around 43 seconds. This value doesn’t reset neither reset signal is high nor enable signal is low.

This global timer would be used as the seed of the randomized number generator. This makes the possibility of the same interval as its previous trial reduce to every 42 seconds.

### Seven-Segments Display (SSD) Driver

The SSD driver have 13 inputs: eight 4-digit Enhanced Binary Coded Decimal (EBCD) number for each digit, one 8-bit dots display expression number, 1kHz clock rising wire, 100MHz system clock, reset and enable. Only 2 outputs are need for this module: one 8-bit output for the digit display with the point and one 8-bit for the digit selector. In the module implementation, the dot output wire separates from the digit expression.

There is a 3-bit register used as the digit selector. This number increases at each 1kHz clock rising edge. The digit selected integer could be calculated with index (displayIndex) with expression ~(8'd1 << displayIndex).

The EBCD is based on the standard Binary Coded Decimal (BCD) and defines some characters for the reaction timer. It expresses a character with a 4-bit unsigned integer. The following table shows the definition of the EBCD.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EBCD | | Glyph | EBCD | | Glyph | EBCD | | Glyph | EBCD | | Glyph |
| 0000 | 0 | 0 | 0100 | 4 | 4 | 1000 | 8 | 8 | 1100 | 12 | A |
| 0001 | 1 | 1 | 0101 | 5 | 5 | 1001 | 9 | 9 | 1101 | 13 | F |
| 0010 | 2 | 2 | 0110 | 6 | 6 | 1010 | 10 | - | 1110 | 14 | I |
| 0011 | 3 | 3 | 0111 | 7 | 7 | 1011 | 11 | blank | 1111 | 15 | L |

The first 10 characters are the same as the normal BCD. Letters ‘A’, ‘F’, ‘I’, ‘L’ are defined for showing the word ‘FAIL’ when user failed to pressed the button within 9.9 seconds. The minus character is used to display the best record number when there is no valid record. Eight minus characters would be shown on SSD.

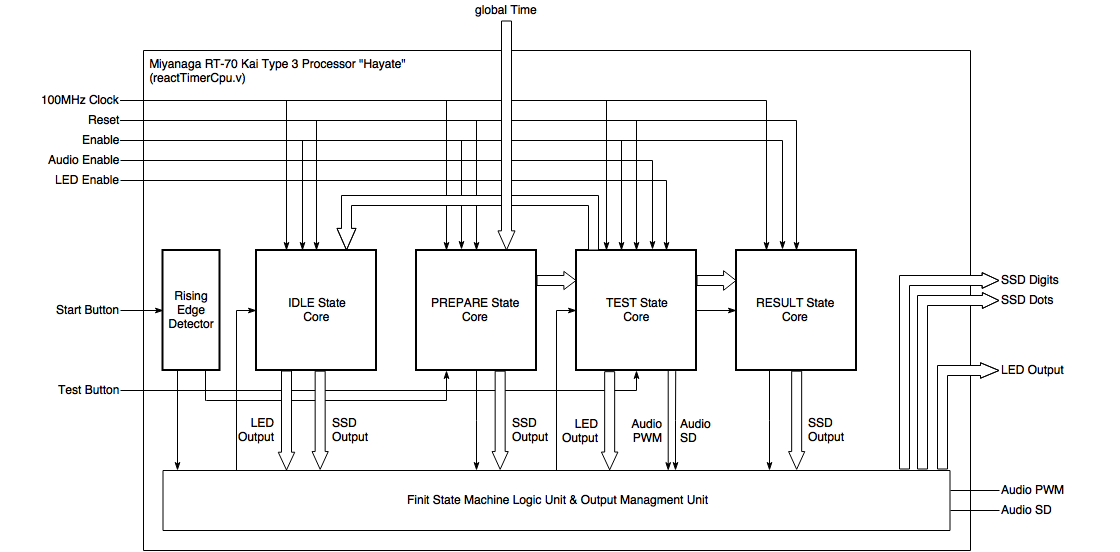
The EBCD code mapping is implemented in the SSD driver module as a single case statement. When the driver is at reset or disable state, it would output blank character with no dot displayed.

## Central Logic Unit (Miyanaga RT-70 Kai Type 3 “Hayate”)

This is the module (**M7T3**) which implements the finite state machine (FSM) of the main reaction timer logic. It contains quad cores inside the module for each state of the FSM and a unified FSM logic switching logic and output management unit. The entire module works as a multiplexer which output the correct signal of the state core module according to the current state of the FSM.

The input signal of **M7T3** are quite simple. Besides the standard 100MHz system clock, reset, enable wires, only four other wires are needed: start test button, test button, audio hint enables and LED hint enables. Outputs are just LED, SSD and audio outputs.

The following diagram shows the inner structure of the **M7T3** processor.



The FSM logic unit inside **M7T3** manages the state switching of the state. There are four states as the specification describes. The responsibility of each state and the policy that transfer between these states are

* IDLE: display the best score and waiting for the start pulse signal.
  + Start button pressed: move to PREPARE state.
  + Or else: play the LED animation.
* PREPARE: generate the random number and show the countdown animation on SSD.
  + Wait both the countdown animation and random number generation complete.
  + Or else: output the random number to TEST state and move to TEST state.
* TEST: according to the random number generate by the PREPARE state, give out the hint signal, wait for user pressing the test button.
  + Play the audio sound hint and/or lighten the LEDs hint after the random number time given by PREPARE state.
  + Test button pressed:
    - If the test button pressed before the hint occurs or not being pressed in 10 seconds, the result is FAIL and move to RESULT state.
    - If the test button pressed in 10 seconds, the result would be recorded and move to RESULT state.
* RESULT: show the test result of the TEST state.
  + After 10 seconds, go to the IDLE state.

If reset signal is on, the state would be back to IDLE state. A 2-bit register is used to store the current state. This register is checked at each 100MHz system clock rising edge by the FSM logic unit. The FSM logic unit is implemented as an *always* block. Within this block, when the state is not changed, the output signal would be refreshed from the output of the state core. If there is no output for one type signal, for instance, there is no audio output for IDLE state, the audio output data would be reset to default value.

### IDLE state core

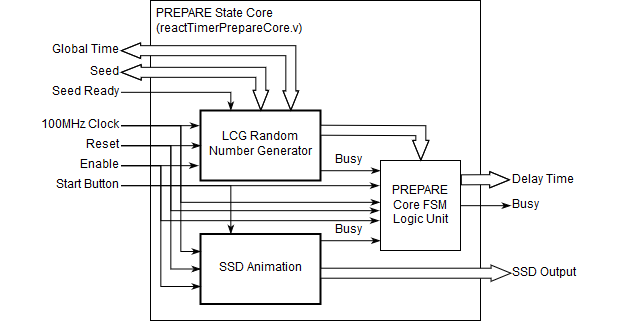
The logic of this core is pretty simple. It stores the binary of the best time. The test result comes directly from the TEST state core. There is also an LED animation playing module embedded in this core.

The outputs of this core are LED animation result and SSD signals.

### PREPARE state core

PREPARE state core implements in a small two state FSM. One state is idle and the other one is busy. For the idle state, this core just waits and detects the start signal rising. The busy state is waiting for the random number generator to generate the random number and SSD animation module to finish playing the countdown animation.

The following diagram shows the PREPARE core architecture.



Once the start button is at rising edge, PREPARE core FSM logic unit would change the state from idle to busy. LCG random number generator and SSD animation module would start to work concurrently. The LCG and SSD animation module both have a busy wire output which shows whether their mission is finished. Once all of these wires changed to 0, this module would change the busy output from 1 to 0.

The output of SSD animation module directly connects to the output of PREPARE core. The busy state of LCG generator is checked by the FSM Logic Unit at every 100MHz clock cycle rising edge. Once the falling edge of the LCG wire is detected, the output 32-bit random number would be modular within 700,000,000 that could guarantee that the waiting seconds are limited to maximum 7 seconds.

#### Random Number Generator

LCG generator is used as the random generator. The equation used in this implementation is . This equation could be completed by Artix-7 within 10ns, hence one system 100MHz clock cycle. The seed is chosen using the global timer counter for each time running the generator.

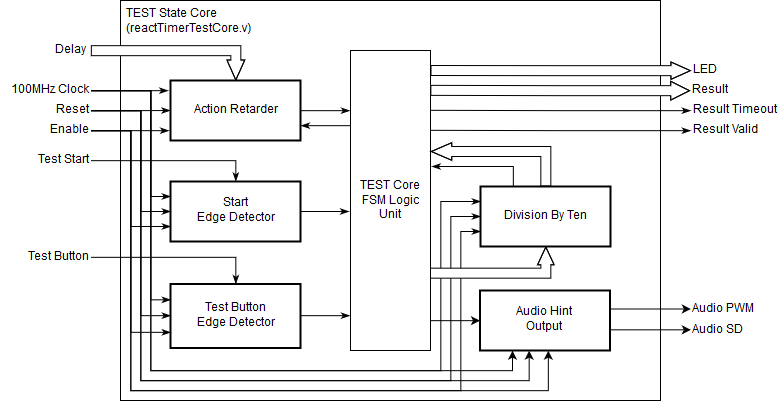
#### SSD Animation

The animation of SSD is implemented by an increase counter which increase 1 at when the timer clock rising. The clock is set to be 1Hz and increase the frame index and update the output frame according to the frame index at the rising edge of system 100MHz clock. The frames are stored as a 32-bit integer array inside the module.

### TEST state core

After the PREPARE core provides the delay duration, the TEST state core start working. This core is responsible for showing the hint after a specific period of time, this is implemented by an action retarder. Then it would measure the gap of the hint provided to the test button pressed. Last, it calculates the data to be output.

The following diagram shows the inner structures of the TEST state core.



The inner framework of the TEST state core is a FSM as well. There are 3 states for this core:

* Idle state: waiting for the test start signal. When the start signal pulse comes, the state switch to wait state. At the same time, the FSM Logic Unit would give action retarder a pulse. The action retarder would give another pulse after the time provided by the PREPARE state core.
* Wait state: At this state, the action retarder would hold the pulse for a while, and output the pulse after several clock cycles, then move to test state. FSM Unit detects this pulse for each 100MHz clock cycle rising edge, and then move to test state. If tester hits the test button at this state, this test would be marked as ‘FAIL’ and move to finish state.
* Test state: According to the configuration, the LED would be shown and/or the audio would be played. At the same time, counter starts to count from 0 and increases 1 for each 100MHz clock cycle. Once the user hits the test button, the counter would stop to count and then move to the finish state. If the counter reaches the limitation (by the default, is 999,999,999, which is actually 9.999 seconds), this test would also be marked as ‘FAIL’, then move to finish state.
* Finish state: the result of the counter is a 9-digit integer in decimal. The number goes into a division module which could divide a number by 10. If the result is not ‘FAIL’, it would wait until the divider finish and then mark the result as valid. Or else set the result to be 0, set the timeout signal as 1, then mark the result as valid.

Action retarder is just a 2-state FSM which detect the input pulse and then wait by increasing a counter as it reaches the number provided by the TEST state core FSM logic unit. It would ignore all the other pulses while holding the first signal it detects.

#### Division by Ten Module

The precise of the counter is 10ns. However, we have only 8 digits which allows us to display the number at 100ns. So we have to divide the counter result by 10. We originally use counter / 10 here to calculate this value. However, it fails to reach the time constraints (provides a negative WNS). Instead of using the division calculation directly, we are using an 8 clock cycles delay divider to do this calculation.

In this algorithm, we need two parameters: and . Suppose the number we want to divide by 10 is . First step, we can get () by

next, get () by

after that, get () by

then, get () by

afterwards, get () by

This number is actually very closed to the data as what we want. To make the data more precise, we need to calculate and check it is greater than 9 or not by

and finally get the result as what we want

which is correct in the 32-bit integer value for division by 10. This module is implemented by a 9-state FSM, which means that it adds a 9-cycle delay to the system at 100MHz. Because