The project, named Octopus, will build a multicore microcontroller consisting of 8 RISC-V 32 bit cores which are running synchronous, each with it's own small RAM, at least 16 KiB per core. The basic concept is like the Propeller (P1) from Parallax which was open sourced in 2014.
(https://www.parallax.com/microcontrollers/propeller-1-open-source)

The RISC-V cores used are of type RV32IMF. The modifications will be, beside the use of RISC-V cores, in the architecture of used RAM. The following description reflects these changes.

In contrast to the original P1 there exists only RAM on each core, which is dual-ported memory to feature shared access by other cores. The shared main RAM found on P1 will be omitted as the access to those is slowly due to the round-robin access if two cores will communicate.

It should be possible to use the available RAM of all cores through different configuration modes much more flexible depending on delevoper requirements.
1) One mode works like the P1, but instead of accessing the shared main RAM the access goes direct to the RAM of another core. One of the RAM ports is used to build access in round-robin fashion. The other port is used by the related core.

2) A second mode allows to configure two cores as pair to use the RAM of both cores mapped together as RAM of doubled size. Hence, you can have up to four core pairs or at least only one pair. As dual-ported RAM is used it is possible to run both cores using the RAM as shared memory without access delay. The round-robin access to the RAM of other cores is possible in this mode, as long the related core is not running in pair mode. Hence, access from other cores to the RAM of the core pair is not possible as all ports of these RAM are already in use.

3) A third mode configures the whole or limited RAM of all cores mapped together as RAM accessible by core 0. If use of limited RAM is configured the remaining RAM can be used by those cores as usual. For example, if you need four times of RAM for core 0, the cores 4 - 7 can be used as usual. The cores 1 - 3 can also be used but needs care with placing the code in the code image to ensure that the code for these cores is in the right range of RAM. Core 0 uses in this mode one RAM port and core 1 - 3 the second port. Hence, it is possible to run core 0 - 3 using the same shared RAM without access delay. Core 0 - 3 can also access the RAM of core 4 - 7 in round-robin fashion. But there is no access possible from those cores to the RAM of core 0 - 3 as all ports of those RAM are already in use.

In general there exists one transmit and one receive channel per core for communication between two cores using duplex or three cores using simplex communication at the same time. This acts as DMA to transfer data from/to RAM between the involved cores without any impact of the running cores. A base address is set by the sender and by the receiver and the size of data to transfer is controlled by sender side. There is a handshake required where the receiver core must enable the transmission. Afterwards the sender is able to

start the transmission. One 32 bit word is transfered within two clocks. The state of the transfer can be checked from parties involved. If a transfer is completed the communication can be reconfigured by the cores. Hence, it will be possible to build a communication between more than two/three cores simultaneously.
Maybe this communication feature is not available for all cores depending from used configuration mode as one RAM port is needed to get access without any impact to the running cores. This needs to be investigated.

These are the important changes. Additional we could add some new features to each core like UARTs or ADC/DAC on GPIOs. But this is not planned yet.