Simulated Modbus Weird Machine Gadgets Turing Completeness:

Samuel Jackson

Our Modbus gadget system is built as a “weird machine” composed of layered abstractions over simulated Modbus Register operations (FC03, FC06) and Coil operations (FC01, FC05). The lowest layer, ProtocolGadget, which exposes basic read and write primitives corresponding to Modbus function codes. The Primitives act as the machine’s memory operations: the registers behave like integer cells (16 bit) while the coils serve as Boolean flags (1 bit):

ModBus Register Operations:

FC03: Read Register

FC06: Write Register

Modbus Coil Operations:

FC01 – Read Coil

FC05 – Write Coil

At the core a system is considered Turing complete if it can (1) store and modify data, (2) make decisions based on the data (branching) and (3) execute operations conditionally or reapeatdy (loops). The gadgets set built off the simulated Modbus protocol satifies these criteria:

1. Memory and State Representation

The Modbus registers act as writable memory cells storing integers in arbitrary range. The write_register, read_register, increment_register, and decrement_register functions implement the basic building blocks of computation and state transition, effectively transforming simple Modbus read/write semantics into memory operations comparable to load, store, and increment instructions on an unbounded tape. Coils provide a binary memory system used for flags and synchronization, similar to Boolean variables.

2. Conditional Logic

The ControlGadget’s conditional_write function and the ConditionalExecutor class implement conditional branching. By comparing register values to thresholds and triggering different write or actions functions, the system supports if / else logic. This mechanism can simulate binary decisions structures (e.g conditional branching or branch instructions)

3. Iteration and Control Flow

The repeat_until and wait_for_condition methods enable loops and synchronization, serving the role of while or for constructs. These loops can repaetedly operate on register values until a conditional is met, providng the iteration required for computational over arbitrary sequences.

4. Composable State Machines

The StateMachine class lets us define named states (such as “Idle”, “Active”, or “Error) and the rules for moving between them. This means we can model how a system changes over time. For example, switching from one mode of operation to another when certain conditions are met, combining these state transitions with conditional checks and register updates, we can build more complex behaviors that go beyond simulated state changes. In theory, by chaining many states together and allowing them to use shared memory (registers), the system can represent any kind of computation, just like a general-purpose computer.

In short, by chaining simple Modbus read/write operations into structured gadgets, we’ve captured the necessary building blocks and have built a Turing complete “weird machine” entirely through industrial control semantics based on the Modbus protocol.