University Of Bristol
School of Computer Science
https://www.cs.bris.ac.uk

Computer Architecture (COMS10015)

Assessed coursework assignment
Encrypt

Note that:

1. This coursework assignment has a 30 percent weighting, i.e., it represents 30 percent of Credit Points (CPs) associated with COMS10015, and is assessed on an individual basis. The submission deadline is 27/11/25.
2. Before you start work, ensure you are aware of *and* adhere to various regulations[a] which govern coursework-based assessment: pertinent examples include those related to academic integrity.
3. There are numerous support resources available, for example:

- via the unit forum, where you can get help and feedback via n-to-m, collective discussion,
- via any lab. and/or drop-in slot(s), where you can get help and feedback via 1-to-1, personal discussion, or
- via the staff responsible for this coursework assignment: although the above are often preferable, you can make contact in-person or online (e.g., via email).

[a] See both the formal regulations at https://www.bristol.ac.uk/academic-quality/assessment/codeonline.html, and also the less formal advice at https://www.bristol.ac.uk/students/support/academic-advice.

1 Introduction

Imagine that two parties $\mathcal{A}$ and $\mathcal{B}$ engage in communication with each other over a public network (e.g., the Internet): a concrete example could be where they represent a web-browser and web-server respectively. Since the n_b-bit messages they communicate will potentially contain security-critical (e.g., identity-, location-, medical-, or finance-related) information, it is important to prevent a third-party $\mathcal{E}$ having access to them. Assuming $\mathcal{A}$ and $\mathcal{B}$ have agreed on some n_k-bit key k before they start communicating, having them use a block cipher[1] to encrypt and decrypt messages would be one approach to satisfying their requirement for secrecy. The idea is that $\mathcal{A}$ encrypts a plaintext message m to form the ciphertext message $c = \textsc{Enc}(k, m)$ which is sent to $\mathcal{B}$. Then, $\mathcal{B}$ decrypts the ciphertext message by computing $m' = \textsc{Dec}(k, c)$ and thereby recovers the same plaintext message, i.e., $m' = m$. This approach is effective because Enc and Dec are carefully designed so that 1) they act as each others inverse under k, and 2) security depends on k alone, not on knowledge of Enc and Dec: even if an attacker $\mathcal{E}$ intercepts c, they cannot easily recover m without also knowing k.

A given block cipher design specifies the algorithms Enc and Dec and associated parameters, e.g., n_k and n_b; numerous such designs exist, the de facto choice being the Advanced Encryption Standard (AES) [1]. AES replaced the Data Encryption Standard (DES) [2], standardised in the 1970s. DES is less efficient in software than AES, which should be no surprise: it was designed in an era when block ciphers were more often implemented in hardware, and uses "hardware friendly" components as a result.

2 Terms and conditions

- The assignment description may refer to the ASCII text file `question.txt`, or more generally "the marksheet": complete and include this file in your submission. This is important, in the sense that 1) it offers *you* clarity with respect to the assessment process, e.g., via a marking scheme, and 2) it offers *us* useful (meta-)information about your submission. Keep in mind that
  - *if* separate *assessment* units exist, they may have different assessment criteria and so marking scheme,
  - the section related to citation of third-party resources includes use of AI: per the University[2] and Faculty[3] guidance, you should "*you should describe and cite your usage [of AI] and quote output [produced by AI] appropriately in your work*".
- Certain aspects of the assignment have a (potentially large) design space of possible approaches. Where there is some debate about the correct or "best" approach, the assignment demands *you* make an informed decision *yourself*: it is therefore not (purely) a programming exercise such that blindly implementing *an* approach will be enough. Such decisions should ideally be based on a reasoned argument formed via your *own* background research (versus relying exclusively on taught content), and clearly documented (e.g., using the marksheet).
- The assignment design includes some heavily supported, closed initial stages which reflect a lower mark, and some mostly unsupported, open later stages which reflects a higher mark. This suggests the marking scale is non-linear: it is clearly easier to obtain X marks in the initial stages than in the final stage. The term open (resp. closed) should be understood as meaning flexibility with respect to options for work, *not* non-specificity with respect to workload: each stage has a clear success criteria that limit the functionality you implement, meaning you can (and should) stop work once they have been satisfied.
- In some, specific instances the required style of Verilog will be dictated by the assignment. If no such requirement exists, however, *you* can select whatever style is appropriate: gate-, RTL-, and behavioural-level Verilog styles are all viable in general. However, whatever style you select, you must consider how your solution relates to real hardware versus purely whether or not it functions correctly in simulation.
- You should submit your work into the correct component via

  https://www.ole.bris.ac.uk

  Include any 1) source code files, 2) text or PDF files, (e.g., documentation) and 3) auxiliary files (e.g., example output), either as required or that *you* feel are relevant. Keep in mind the following points:
  - *If* separate *teaching* and *assessment* units exist, you should submit via the latter *not* the former.
  - Make sure you have actually *made* a submission, rather than saved a draft ready *for* submission; ensure said submission matches what you expect, e.g., by (re)downloading and checking the content.
  - Your *last* submission will be the one assessed, meaning, e.g., you cannot partially *or* entirely "roll-back" to some earlier submission.

[1] https://en.wikipedia.org/wiki/Block_cipher
[2] https://www.bristol.ac.uk/students/support/academic-advice/academic-integrity
[3] https://www.ole.bris.ac.uk/bbcswebdav/pid-8241705-dt-content-rid-48627612_3/xid-48627612_3

- To make the submission process easier, the recommended approach is to develop your solution within the *same* directory structure as the material provided. This will allow you to first create then submit a *single* archive (e.g., `solution.zip` using `zip`, or `solution.tar.gz` using `tar` and `gzip`) of your entire solution, rather than *multiple* separate files.
- Any implementations produced as part of the assignment will be assessed using a platform equivalent to the MVB Linux lab(s). (e.g., MVB-1.15 or MVB-2.11). As such, they *must* compile, execute, and be thoroughly tested using both the operating system and development tool-chain versions available by default.
- Although you can *definitely* expect to receive a partial mark for a partial solution, it will be assessed *as is*. This means 1) there will be no effort to enable either optional or commented functionality (e.g., by uncommenting it, or via specification of compile-time or run-time parameters), and 2) submitting multiple variant solutions is strongly discouraged, but would be dealt with by considering the variant which yields the highest single mark.

3 Description

3.1 Material

Download and unarchive the file

https://assets.phoo.org/COMS10015_2025_TB-4/csdsp/cw/Encrypt/question.tar.gz

somewhere secure in your file system: from here on, we assume `${ARCHIVE}` denotes a path to the resulting, unarchived content illustrated by Figure 1. In particular, you should find

- `question.txt`†, the marksheet mentioned in the assessment terms and conditions,
- `Makefile`, a GNU `make` based build system described in more detail by Appendix C.
- `params.h`, a header file that provides a symbolic definition of parameters such as n_k, n_b, and n_r,
- `vectors_k.txt`, `vectors_m.txt` and `vectors_c.txt`, ASCII text files representing the test vectors described in more detail by Appendix D,
- `encrypt_comb.v`†, `encrypt_iter.v`†, and `encrypt_pipe.v`†, incomplete implementations of modules called `encrypt_comb`, `encrypt_iter`, and `encrypt_pipe`,
- `clr_28bit.v`†, an incomplete implementation of a module called `clr_28bit`,
- `key_schedule.v`†, an incomplete implementation of a module called `key_schedule`,
- `round.v`†, an incomplete implementation of a module called `round`,
- `util.v`, a set of pre-defined support modules including
  - `split_0`, `split_1`, and `split_2`, modules that split a single input into multiple outputs,
  - `merge_0`, `merge_1`, and `merge_2`, modules that merge multiple inputs into a single output,
  - `perm_IP`, `perm_FP`, `perm_E`, `perm_P`, `perm_PC1`, and `perm_PC2`, implementations of the DES permutations, and
  - `sbox_0` through to `sbox_7`, implementations of the DES S-boxes.

plus a set of test stimuli: for a module `X` as defined in `X.v`, the corresponding test stimulus is `X_test` as defined in `X_test.v`. Viewed at face value, that is a *lot* of files! However, it is vital to understand that you can complete the assignment by altering *only* those files marked with a † symbol. Because you do not need to, you should not alter any *other* files: if you *do*, those alterations will be reverted before (and so therefore ignored during) the marking process.

3.2 Overview

Consider an example scenario, where you join the development team for a device $\mathcal{T}$; to address the challenge of secure communication, $\mathcal{T}$ integrates and makes use of a hardware implementation of DES. This assignment models aspects of the scenario outlined above, using Verilog as a vehicle to do so. More specifically, it tasks you with implementing the Enc algorithm for DES in Verilog. Remember that, in essence, Verilog simply offers a neat way to express and experiment with (i.e., simulate) a design you could also write down on paper and reason about in theory: a sensible strategy throughout is to establish understanding "on paper" before then applying it "in practice" (e.g., via source code).

Selection of DES[4] implies $n_k = 64$ and $n_b = 64$, meaning a 64-bit cipher key k is used to encrypt a 64-bit plaintext message m into a 64-bit ciphertext message c. DES is an iterative block cipher, meaning that a full encryption

[4] An accessible introduction to DES is provided by https://en.wikipedia.org/wiki/Data_Encryption_Standard for example. Keep in mind, however, that various technical details relating to DES are out of scope given the task at hand: where that is the case, we simply ignore

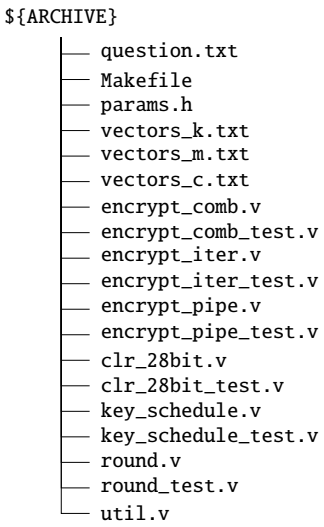


Figure 1: *A diagrammatic description of the material in* `question.tar.gz`.

operation involves successively applying partial encryption rounds (or steps): Figure 2 illustrates this using a block diagram, noting that a total of $n_r = 16$ rounds (numbered 0 to 15 inclusive) are followed (resp. preceded) by a post-processing (resp. pre-processing) step.

3.3 Detail

Stage 1. The goal of this stage is to implement modules that, when combined, act to support some i-th round of encryption. The idea is that by using these modules, the subsequent stages can then explore different implementation strategies for a full encryption operation.

(a) The `clr_28bit` module implements what can be described as a "controlled" left-rotate operation: given a 28-bit `x` and 4-bit `y` as input, it computes

$$\mathtt{r} = \mathtt{x} \lll f(\mathtt{y})$$

as output, where

$$f(\mathtt{y}) = \begin{cases} 1 & \text{if } \mathtt{y} \in \{0, 1, 8, 15\} \\ 2 & \text{otherwise} \end{cases}$$

Put another way, it left-rotates `x` by either 1 or 2 bits depending on `y`.

Complete the module implementation, using Appendix D to verify (as far as possible) that it functions as expected: your implementation should express f as a (set of) Boolean expressions, e.g., using a gate-level Verilog style.

(b) With reference to Figure 2, the `key_schedule` module implements some i-th round of the key schedule: the required implementation is described in a diagrammatic form by Figure 3.

Complete the module implementation, using Appendix D to verify (as far as possible) that it functions as expected.

(c) With reference to Figure 2, the `round` module implements some i-th round of encryption: the required implementation is described in a diagrammatic form by Figure 4.

Complete the module implementation, using Appendix D to verify (as far as possible) that it functions as expected.

Stage 2. The `encrypt_comb` module accepts

- a 64-bit value called `k`, a cipher key, and
- a 64-bit value called `m`, a plaintext message,

as input, and produces

- a 64-bit value called `c`, a ciphertext message,

as output: as such, it computes a full encryption operation.

them. For example, note that DES represents a specific instance of a more general block cipher design principle, i.e., a Feistel network. Also note that only 56 of the 64 bits in k are actually used for encryption; the others are discarded after inclusion in a parity check.

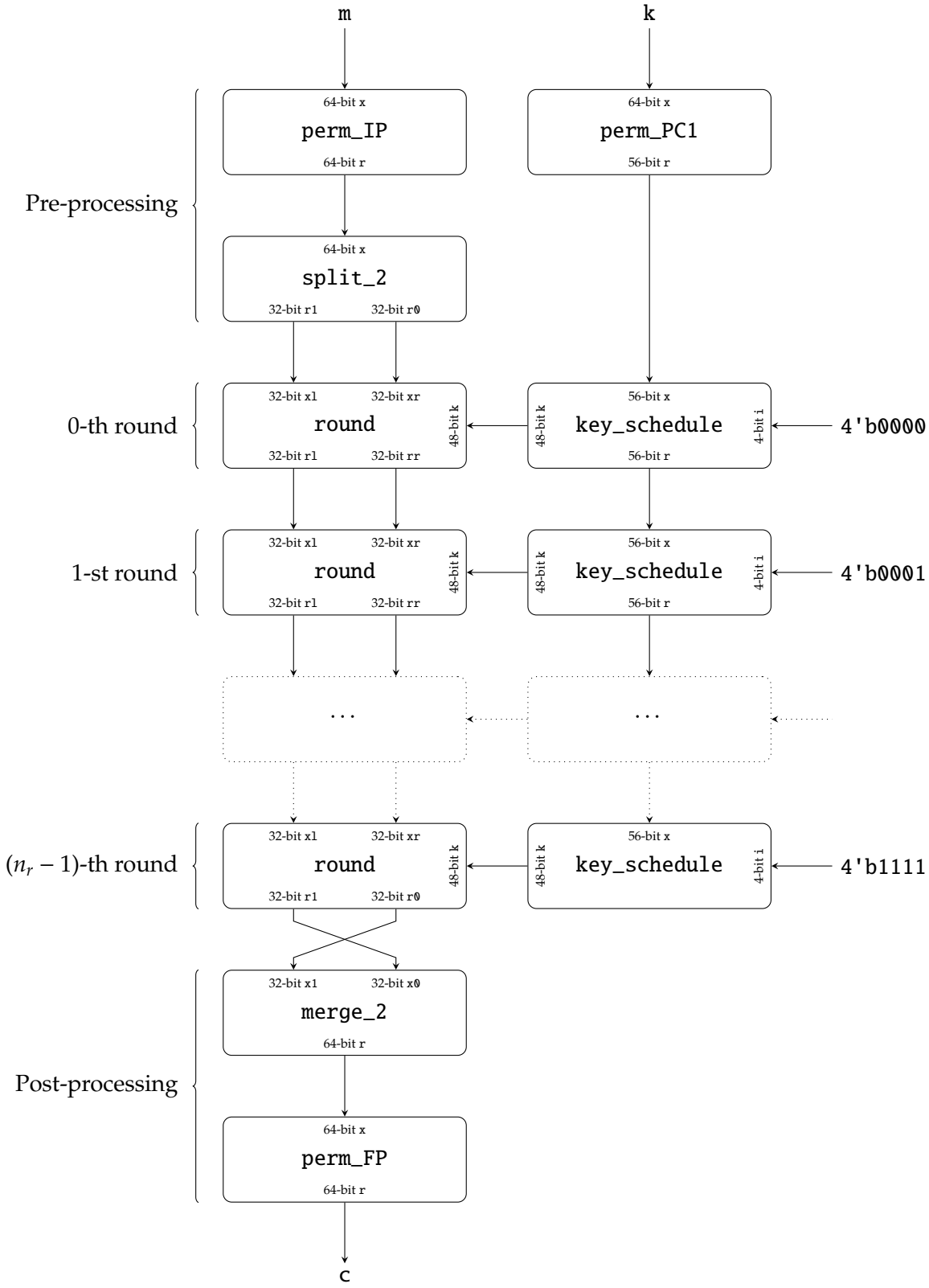


Figure 2: *A block diagram illustrating a full encryption operation using DES, i.e., computation of* $\mathtt{c} = \textsc{Enc}(\mathtt{k}, \mathtt{m})$.

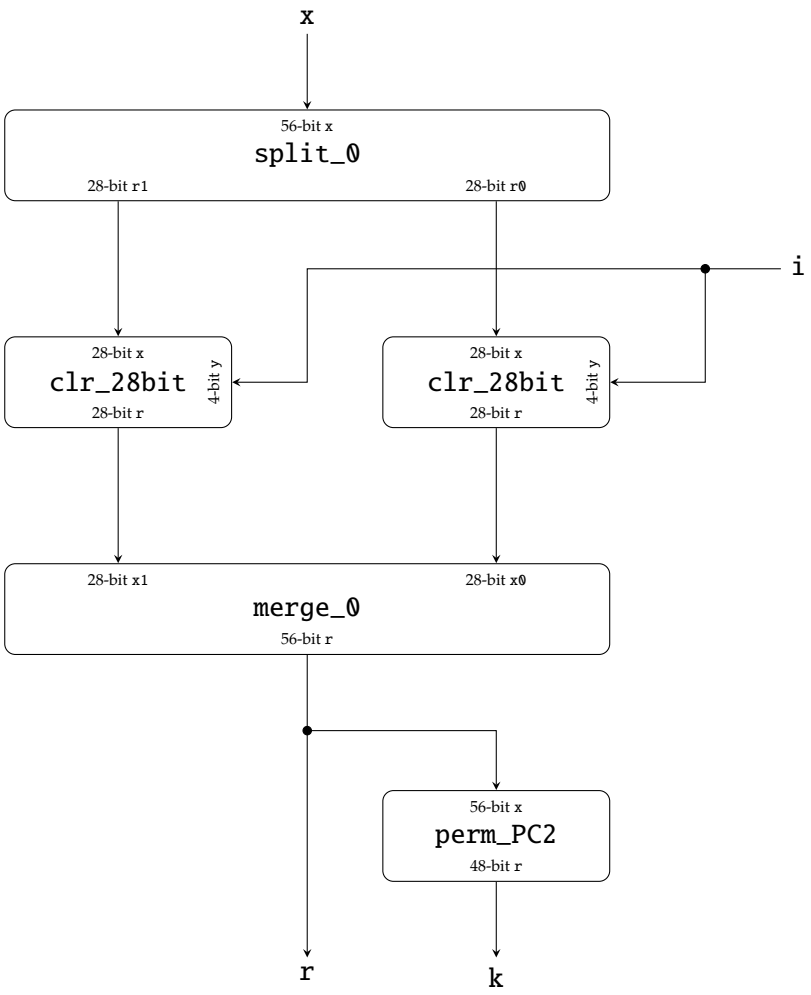


Figure 3: *The* `key_schedule` *module, as used to form the i-th round of DES.*

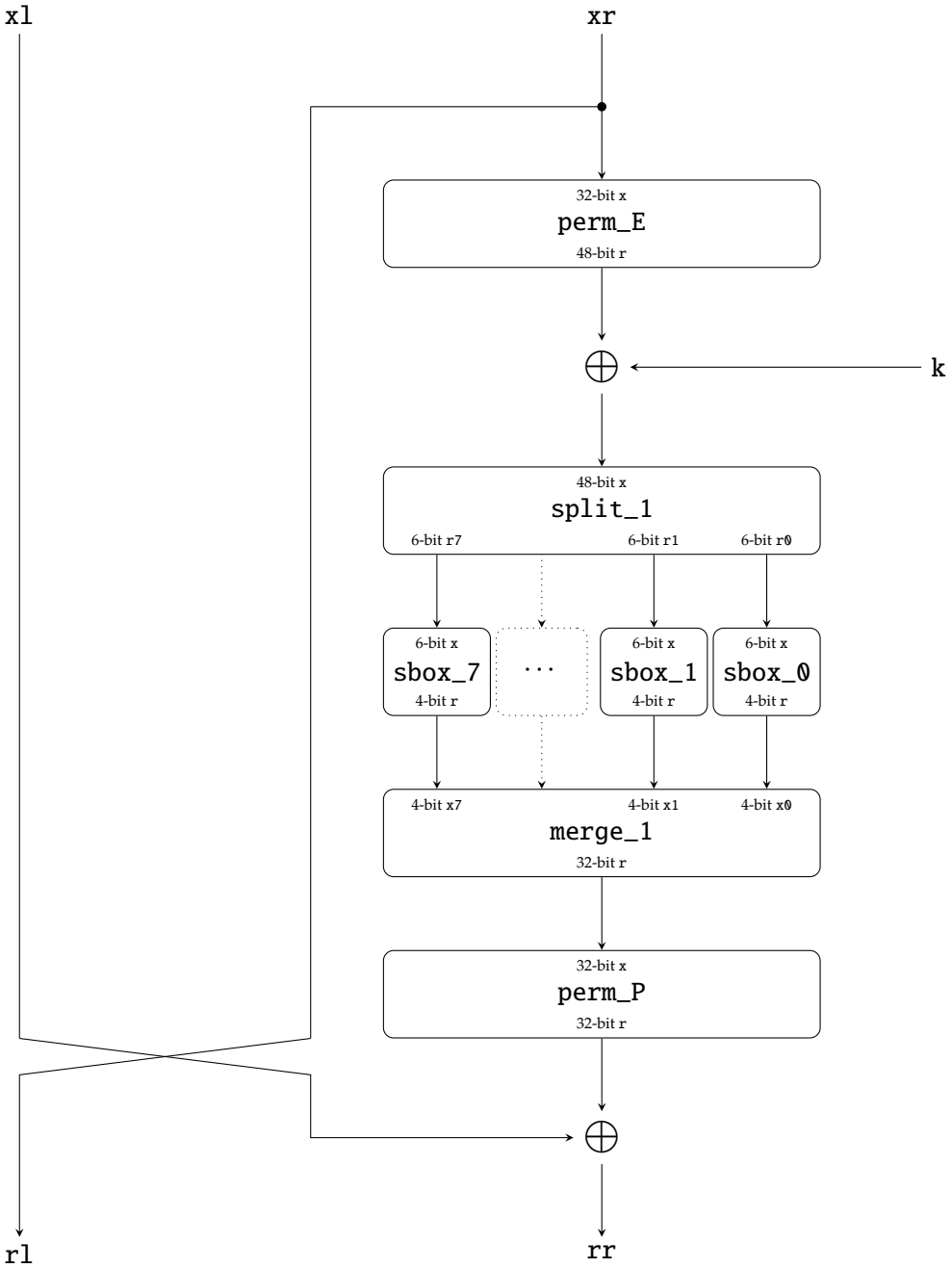


Figure 4: *The* `round` *module, as used to form the i-th round of DES.*

The goal of this stage is to implement the `encrypt_comb` module, using a combinatorial design strategy which replicates Figure 2 in a fairly direct manner. Adopting this strategy trades-off higher area (since n_r rounds are instantiated) in favour of lower latency (since the number of clock cycles for each encryption is 1), in relative terms, and makes the resulting implementation easier to use: the output is computed continuously from the input.

Complete the module implementation, using Appendix D to verify (as far as possible) that it functions as expected.

Stage 3. The `encrypt_iter` module has an interface matching that of `encrypt_comb` except for some additional inputs and outputs, namely

- a 1-bit value called `clk`, a clock signal,
- a 1-bit value called `rst`, a reset signal,
- a 1-bit value called `req`, a request signal, and
- a 1-bit value called `ack`, an acknowledge signal,

and also computes a full encryption operation.

The goal of this stage is to implement the `encrypt_iter` module, using a different, iterative design strategy. Adopting this strategy trades-off higher latency (since the number of clock cycles for each encryption is $\sim n_r$), in favour of lower area (since 1 round is instantiated), in relative terms, and makes the resulting implementation harder to use: a control protocol outlined by Appendix A must be followed to provide input and accept output correctly.

Complete the module implementation, using Appendix D to verify (as far as possible) that it functions as expected.

Advice. The natural way to design and implement the control protocol is by treating it as a Finite State Machine (FSM). More so than other stages, it is important to explain the design of said FSM: use either `question.txt` and/or comments in your source code to do so.

Advice. Until you implement the module, simulating it will "hang" as a result of incorrect interaction with the test stimulus. In more detail, the test stimulus follows the control protocol and hence waits for the module to set `ack` to 1 at the end of computation: `ack` is not updated by the incomplete implementation, so the test stimulus ends up waiting forever.

Stage 4. The `encrypt_pipe` module has an interface matching that of `encrypt_comb` except for some additional inputs and outputs, namely

- a 1-bit value called `clk`, a clock signal, and
- a 1-bit value called `rst`, a reset signal,

and also computes a full encryption operation.

The goal of this stage is to implement the `encrypt_pipe` module, using a different, pipelined design strategy which delivers a different trade-off: this strategy focuses on maximising throughput, rather than minimising latency (as with `encrypt_comb`) or area (as with `encrypt_iter`). Note that a control protocol outlined by Appendix B must be followed to provide input and accept output correctly.

Complete the module implementation, using Appendix D to verify (as far as possible) that it functions as expected.

Advice. Before investing significant effort in design or implementation tasks, it is crucial you first conduct some background research in order to understand the concept of pipelining.

Advice. In general, the number of pipeline stages represents a parameter for which a concrete value must be selected. Here, however, the assumption is that a fully pipelined design strategy is employed. This implies 1) there are n_r pipeline stages, so 2) computation of an output from the associate inputs has an n_r clock cycle latency.

References

[1] *Advanced Encryption Standard (AES)*. National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) 197. 2001. URL: https://doi.org/10.6028/NIST.FIPS.197 (see p. 2).

[2] *Data Encryption Standard (DES)*. National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) 46-3. 1999 (see p. 2).

A Control protocol for the iterative design strategy

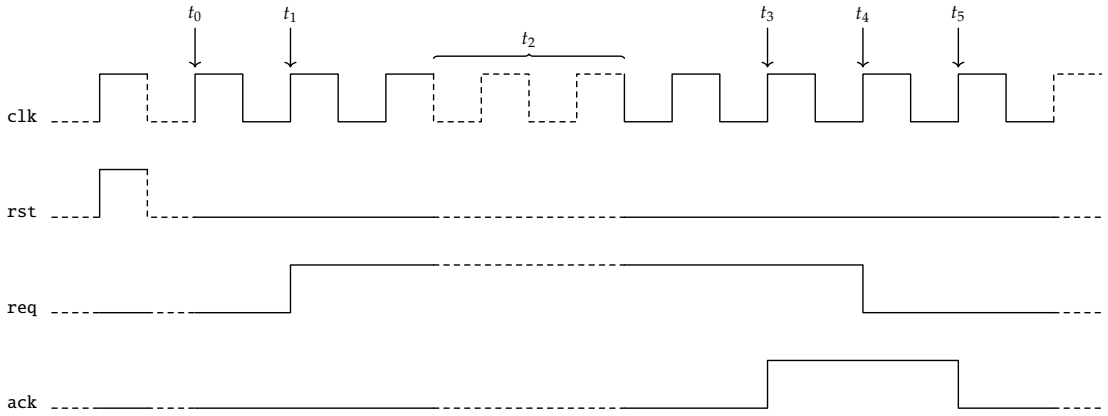
A.1 Problem

The iterative design strategy requires an understanding of how the module (i.e., `encrypt_iter`) and the user *of* said module (e.g., the test stimulus `encrypt_iter_test`) interact. Two underlying problems exist, namely 1) the module does not know know when to start computation (i.e., when input is available), and 2) the user does not know know when computation has finished (i.e., when output is available).

The solution of both problems is for both parties to follow a protocol: this is based on use of the request signal `req` and acknowledge signal `ack`, and driven by (positive edges of) the clock signal `clk`. You could frame interaction between the module and user as communication between (i.e., of input and output to and from) them, and so a "conversation" controlled (or structured) by the protocol: the rules of said protocol essentially mean, e.g., one party cannot be "confused" as a result of communication by the other.

A.2 Protocol

Based on the following diagrammatic example



the (intentionally simple) protocol can be explained as follows, noting the initial toggling of `rst` from 0 to 1 and back again signals a reset (e.g., of any registers to an initial state):

- At some positive edge on `clk` (labelled t_0), both `req` and `ack` are initially 0.
- At some positive edge on `clk` (labelled t_1), the user wants the module start a computation. It proceeds by 1) driving values onto any inputs (i.e., `k` and `m`), then 2) changing `req` from 0 to 1.
- The module notices the change to (e.g., positive edge on) `req`, and concludes that the inputs are available. Note that, in general, it would need to store the inputs ready for subsequent use. The reason for storing them internally within the module stems from a need to be pessimistic: the module must pessimistically assume any externally provided input *may* be changed *during* computation which uses them. However, given the assignment remit, we relax the requirement to do so by guaranteeing all inputs are stable throughout the computation (i.e., they will not change until the computation is complete, so there is no need to store them internally).
- During some period (labelled t_2), the module computes the outputs from the inputs using `clk` to trigger each constituent step; during this period, the user is essentially waiting for the computation to finish.
- At some positive edge on `clk` (labelled t_3), the module finishes the computation. It proceeds by 1) driving values onto any outputs (i.e., `c`), then 2) changing `ack` from 0 to 1.
- The user notices the change to (e.g., positive edge on) `ack`, and concludes that the outputs are available. It proceeds by 1) storing any outputs ready for subsequent use, then 2) changing `req` from 1 to 0.
- The module notices the change to (e.g., negative edge on) `req`, and concludes that the interaction is finished. It proceeds by changing `ack` from 1 to 0.
- The user notices the change to (e.g., negative edge on) `ack` and concludes that the interaction is finished.
- Since both `req` and `ack` are 0 again, the module and user are ready to engage in successive interactions if/when need be.

B Control protocol for the pipelined design strategy

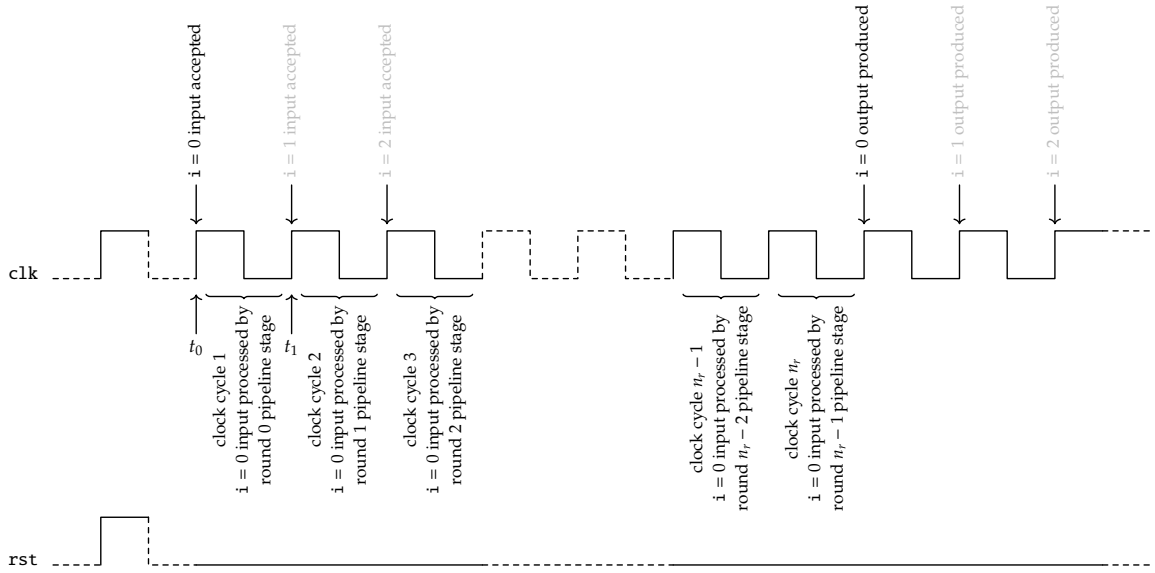
B.1 Problem

In the same way as described in Chapter A, the pipelined design strategy requires careful control (or structure) with respect to interaction between the module and user; this is realised by having them adhere to a protocol.

Similar underlying problems exist, but they now stem from the fact that, at a given instant, a pipeline with n_r stages could be described as simultaneously processing n_r independent computations, each of which is at a different stage of completeness i.e., less (resp. more) complete within initial (resp. latter) stages. This description implies a need to control what the pipeline does, and when it does it, that that, e.g., inputs and outputs are accepted and produced correctly, and computations of one from the other progresses correctly.

B.2 Protocol

Based on the following diagrammatic example



the (intentionally simple) protocol can be explained as follows, noting the initial toggling of `rst` from 0 to 1 and back again signals a reset (e.g., of any registers to an initial state): at every positive edge on `clk`,

- the module accepts input (i.e., `k` and `m`) from the user and
- the user accepts output (i.e., `c`) from the module.

Despite being (or perhaps because it is) so simple, several subtle but important points need to be considered: these points are reflected by the test stimulus, but warrant discussion and explanation. First, the output associated with a specific input will be spaced n_r clock cycles apart; this fact stems from the number of stages in and hence computational latency of the pipeline. Second, the output will be invalid in some clock cycles; this fact stems from it not corresponding to any value associated input. For example, the 0-th input is provided to the pipeline on the clock edge labelled t_0. On the clock edge labelled t_1, that input have only been processed by 1 pipeline stage: this means the associated output is not ready, and so what the pipeline produces is therefore is invalid (because *no* input has been fully processed at that instant). Third, `clk` is now the *only* form of synchronisation between the module and user: in contrast to Chapter A, for example, `req` and `ack` no longer exist. This places a stricter constraint on both the module and user, in the sense that the input (resp. output) must be ready at the required positive edge on `clk` (because there is no way to wait, e.g., if the input (resp. output) is *not* ready).

C Developing your solution

C.1 Workflow

To use the content provided, and thus support development of your solution based on it, you can and so should adopt the same approach as presented in the lab. worksheet(s). Before you start:

1. update your `${PATH}`[5] environment variable by executing

   ```
   export PATH="${PATH}:/opt/anaconda/3-2025/bin
   ```

2. check said update worked correctly by executing

   ```
   which iverilog
   which gtkwave
   ```

   noting that any reported error (e.g., `no iverilog in ...` or similar) suggests it did not: ask for help!

Imagine you have 1) `X.v`, which implements a module `X`, plus 2) `X_test.v`, which implements a module `X_test`, where the latter acts as a test stimulus for the former. Although all of the steps related to use of `X.v` and `X_test.v` could be performed manually, the `Makefile` provided represents an automated build system which implies less effort *and* less chance of error. Based on the use of `Makefile`, an edit-compile-execute style design cycle can be roughly summerised as follows:

1. Edit the Verilog source code file `X.v`.
2. Execute

   ```
   make clean
   ```

   to clean (i.e., remove) residual files stemming from previous compilation or simulation steps.
3. Execute

   ```
   make X_test.vvp
   ```

   to compile the design using `iverilog`: doing so combines the Verilog source code file `X.v` with the associated test stimulus `X_test.v` to produce the executable `X_test.vvp`.
4. Execute

   ```
   make X_test.vcd
   ```

   to simulate the design using `vvp`: doing so produces 1) some machine-readable output via a Value Change Dump (VCD)[6] file `X_test.vcd`, plus 2) some human-readable output via the terminal.

   Not all test stimuli are fully-automatic; some require arguments (or parameters) to control them. In such a case, `Makefile` uses the `ARGS` environment variable to capture arguments it then passes to `vvp`. For example, imagine `X_test` requires three 1-bit arguments called `x`, `y`, and `z`: instead of the above, one could instead execute

   ```
   make ARGS="+x=0 +y=1 +z=0" X_test.vcd
   ```

   to set `x` = 0, `y` = 1, and `z` = 0.
5. Execute

   ```
   gtkwave X_test.vcd
   ```

   to visualise the resulting VCD file using `gtkwave`.

Once you have completed your solution, execute

```
make solution.tar.gz
```

to automatically create a single archive, i.e., the file, `solution.tar.gz`, consisting of *all* files in the current working directory.

C.2 Example

In more concrete terms, one might consider the case where `X = encrypt_comb` which means that the Verilog source code files `encrypt_comb.v` and `encrypt_comb_test.v` describe the modules `encrypt_comb`, and `encrypt_comb_test` respectively. The development workflow would then be

1. edit `encrypt_comb.v` to complete the module implementation,

[5] http://en.wikipedia.org/wiki/PATH_(variable)
[6] https://en.wikipedia.org/wiki/Value_change_dump

2. execute `make clean`,
3. execute `make encrypt_comb_test.vvp`,
4. execute `make encrypt_comb_test.vcd`,
5. execute `gtkwave encrypt_comb_test.vcd`.

D Testing your solution

D.1 Test vectors

The process of testing and debugging an implementation of some arithmetic operation (e.g., a ripple-carry adder) is arguably made easier by the fact we can (and sometimes do) compute the same results manually; this fact affords some intuition about whether the correct result is produced, and, crucially, the reason why if not. The same is not true of a block cipher implementation, where, by design, there is no analogous intuition for what the correct result should be (in the sense it should "look" random).

This problem demands a considered approach to testing and debugging, for which a number of number of different options exist. For example:

1. Given an implementation of only Enc, we can test whether $c \stackrel{?}{=} \textsc{Enc}(k, m)$ if provided with k and m *and* the corresponding, known to be correct c.
2. Given an implementation of both Enc and Dec, we can apply a consistency check by testing whether $m \stackrel{?}{=} \textsc{Dec}(k, \textsc{Enc}(k, m))$ for some random k and m.

Each has advantages and disadvantages, and one might sensibly argue that a combination of these (and others) would be ideal. In the context of this assignment we focus on the former, which is based on availability of a test vector[7], i.e., a set of inputs and expected outputs: we test an implementation involved by providing it the inputs then comparing the output *it* computes against the one expected. The test stimuli `encrypt_comb_test.v`, `encrypt_iter_test.v`, and `encrypt_pipe_test.v` provided use exactly this approach: each i-th line of the ASCII text file

- `vectors_k.txt` contains $k[i]$, the i-th 64-bit hexadecimal cipher key,
- `vectors_m.txt` contains $m[i]$, the i-th 64-bit hexadecimal plaintext message, and
- `vectors_c.txt` contains $c[i]$, the i-th 64-bit hexadecimal ciphertext message,

such that $c[i] = \textsc{Enc}(k[i], m[i])$. For example, the first lines contain

$$\begin{array}{rcrcl} k[0] & = & \mathit{FEDCBA9876543210}_{(16)} & \mapsto & \texttt{64'hFEDCBA9876543210} \\ m[0] & = & \mathit{30323231534D4F43}_{(16)} & \mapsto & \texttt{64'h30323231534D4F43} \\ c[0] & = & \mathit{ABD3787AC6026CB1}_{(16)} & \mapsto & \texttt{64'hABD3787AC6026CB1} \end{array}$$

A given test stimuli loads this content into a corresponding memory using the `readmemh` system task, then tests whether $c[i] \stackrel{?}{=} \textsc{Enc}(k[i], m[i])$ for each i-th test vector. This allows the test process to be automatic, and avoids the test stimuli itself becoming too verbose (e.g., due to expression of the test vectors as Verilog source code). Note that displayed output from a test stimuli of this type, as produced by the `$display` system task, has a standard format:

- A line of the form `![<index>] <string>` (e.g., `![0] aborting (ack = 1'bZ)`), describes an error, e.g., the simulation and hence test process aborted for some cited reason.
- A line of the form `>[<index>] <signal>=<value>` (e.g., `>[0] k=dc8770e93ea141e1fc67`), describes an input, i.e., that for vector number `index` the (input) signal `signal` has the value `value`.
- A line of the form `<[<index>] <signal>=<value>` (e.g., `<[0] c=02debc8cb87bc942`), describes an output, i.e., that for vector number `index` the (output) signal `signal` has the value `value`.
- A line of the form `?[<index>] pass` (e.g., `?[0] pass`), describes a pass test outcome, i.e., that for vector number `index` all of the computed outputs matched the associated expected output.
- A line of the form `?[<index>] fail` (e.g., `?[0] fail`), describes a fail test outcome, i.e., that for vector number `index` one of the computed outputs did not match the associated expected output.

D.2 Fully worked example

Although the test vectors described above are useful for testing whether an *overall* result is correct, when said result is *in*correct they offer little or no insight into what or where the problem might be. To help address this limitation, the following represents a fully worked example for test vector $i = 0$. Note that it includes all intermediate inputs and outputs for every operation within every round: although this makes it extremely verbose, it should, e.g., allow you identify exactly where your implementation and the example differ.

[7] https://en.wikipedia.org/wiki/Test_vector

Input

```
k    =  64'hfedcba9876543210
m    =  64'h30323231534d4f43
```

Pre-processing

- perm_IP
```
x    =  64'h30323231534d4f43
r    =  64'hf01f60f8000f60d6
```
- split_2
```
x    =  64'hf01f60f8000f60d6
r0   =  32'h000f60d6
r1   =  32'hf01f60f8
```
- perm_PC1
```
x    =  64'hfedcba9876543210
r    =  56'h0f3355f55330ff
```

0-th round

- round
```
xl   =  32'hf01f60f8
xr   =  32'h000f60d6
k    =  48'hf4fd9864b65a
rl   =  32'h000f60d6
rr   =  32'h3c77eacc
```
  - perm_E
```
x    =  32'h000f60d6
r    =  48'h00005eb016ac
```
  - split_1
```
x    =  48'hf4fdc6d4a0f6
r0   =  6'h36
r1   =  6'h03
r2   =  6'h0a
r3   =  6'h35
r4   =  6'h06
r5   =  6'h37
r6   =  6'h0f
r7   =  6'h3d
```
  - sbox_0
```
x    =  6'h36
r    =  4'hd
```
  - sbox_1
```
x    =  6'h03
r    =  4'h0
```
  - sbox_2
```
x    =  6'h0a
r    =  4'h2
```
  - sbox_3
```
x    =  6'h35
r    =  4'h0
```
  - sbox_4
```
x    =  6'h06
r    =  4'h3
```
  - sbox_5
```
x    =  6'h37
r    =  4'h3
```
  - sbox_6
```
x    =  6'h0f
r    =  4'he
```

- sbox_7
```
x    =  6'h3d
r    =  4'h6
```
- merge_1
```
x0   =  6'hd
x1   =  6'h0
x2   =  6'h2
x3   =  6'h0
x4   =  6'h3
x5   =  6'h3
x6   =  6'he
x7   =  6'h6
r    =  48'h6e33020d
```
- perm_P
```
x    =  32'h6e33020d
r    =  32'hcc688a34
```
- key_schedule
```
x    =  56'h0f3355f55330ff
i    =  4'h0
r    =  56'h1e66abeaa661fe
k    =  48'hf4fd9864b65a
```
  * split_0
```
x    =  56'h0f3355f55330ff
r0   =  28'h55330ff
r1   =  28'h0f3355f
```
  * clr_28bit (left-hand instance)
```
x    =  28'h0f3355f
y    =  4'h0
r    =  28'h1e66abe
```
  * clr_28bit (right-hand instance)
```
x    =  28'h55330ff
y    =  4'h0
r    =  28'haa661fe
```
  * merge_0
```
x0   =  28'haa661fe
x1   =  28'h1e66abe
r    =  56'h1e66abeaa661fe
```
  * perm_PC2
```
x    =  56'h1e66abeaa661fe
r    =  48'hf4fd9864b65a
```

1-st round

- round
```
xl   =  32'h000f60d6
xr   =  32'h3c77eacc
k    =  48'h9659a6da95d9
rl   =  32'h3c77eacc
rr   =  32'h72731955
```
- perm_E
```
x    =  32'h3c77eacc
r    =  48'h1f83aff55658
```
- split_1

```
x    =  48'h89da092fc381
r0   =  6'h01
r1   =  6'h0e
r2   =  6'h3c
r3   =  6'h0b
r4   =  6'h09
r5   =  6'h28
r6   =  6'h1d
r7   =  6'h22
```

- sbox_0

```
x    =  6'h01
r    =  4'h1
```

- sbox_1

```
x    =  6'h0e
r    =  4'hd
```

- sbox_2

```
x    =  6'h3c
r    =  4'hb
```

- sbox_3

```
x    =  6'h0b
r    =  4'h7
```

- sbox_4

```
x    =  6'h09
r    =  4'h6
```

- sbox_5

```
x    =  6'h28
r    =  4'h8
```

- sbox_6

```
x    =  6'h1d
r    =  4'hb
```

- sbox_7

```
x    =  6'h22
r    =  4'h1
```

- merge_1

```
x0   =  6'h1
x1   =  6'hd
x2   =  6'hb
x3   =  6'h7
x4   =  6'h6
x5   =  6'h8
x6   =  6'hb
x7   =  6'h1
r    =  48'h1b867bd1
```

- perm_P

```
x    =  32'h1b867bd1
r    =  32'h727c7983
```

- key_schedule

```
x    =  56'h1e66abeaa661fe
i    =  4'h1
r    =  56'h3ccd57c54cc3fd
k    =  48'h9659a6da95d9
```

  * split_0

```
x    =  56'h1e66abeaa661fe
r0   =  28'haa661fe
r1   =  28'h1e66abe
```

  * clr_28bit (left-hand instance)

```
x    =  28'h1e66abe
y    =  4'h1
r    =  28'h3ccd57c
```

  * clr_28bit (right-hand instance)

```
x    =  28'haa661fe
y    =  4'h1
r    =  28'h54cc3fd
```

* merge_0

```
x0   =  28'h54cc3fd
x1   =  28'h3ccd57c
r    =  56'h3ccd57c54cc3fd
```

* perm_PC2

```
x    =  56'h3ccd57c54cc3fd
r    =  48'h9659a6da95d9
```

2-nd round

- round

```
xl   =  32'h3c77eacc
xr   =  32'h72731955
k    =  48'hba2b754bd72d
rl   =  32'h72731955
rr   =  32'hf4bb5951
```

  - perm_E

```
x    =  32'h72731955
r    =  48'hba43a68f2aaa
```

  - split_1

```
x    =  48'h0068d3c4fd87
r0   =  6'h07
r1   =  6'h36
r2   =  6'h0f
r3   =  6'h31
r4   =  6'h13
r5   =  6'h23
r6   =  6'h06
r7   =  6'h00
```

  - sbox_0

```
x    =  6'h07
r    =  4'h8
```

  - sbox_1

```
x    =  6'h36
r    =  4'h8
```

  - sbox_2

```
x    =  6'h0f
r    =  4'h5
```

  - sbox_3

```
x    =  6'h31
r    =  4'h6
```

  - sbox_4

```
x    =  6'h13
r    =  4'h7
```

  - sbox_5

```
x    =  6'h23
r    =  4'ha
```

  - sbox_6

```
x    =  6'h06
r    =  4'he
```

  - sbox_7

```
x    =  6'h00
r    =  4'he
```

  - merge_1

```
x0   =   6'h8
x1   =   6'h8
x2   =   6'h5
x3   =   6'h6
x4   =   6'h7
x5   =   6'ha
x6   =   6'he
x7   =   6'he
r    =   48'heea76588
```

- perm_P

```
x    =   32'heea76588
r    =   32'hc8ccb39d
```

- key_schedule

```
x    =   56'h3ccd57c54cc3fd
i    =   4'h2
r    =   56'hf3355f05330ff5
k    =   48'hba2b754bd72d
```

  * split_0

```
x    =   56'h3ccd57c54cc3fd
r0   =   28'h54cc3fd
r1   =   28'h3ccd57c
```

  * clr_28bit (left-hand instance)

```
x    =   28'h3ccd57c
y    =   4'h2
r    =   28'hf3355f0
```

  * clr_28bit (right-hand instance)

```
x    =   28'h54cc3fd
y    =   4'h2
r    =   28'h5330ff5
```

  * merge_0

```
x0   =   28'h5330ff5
x1   =   28'hf3355f0
r    =   56'hf3355f05330ff5
```

  * perm_PC2

```
x    =   56'hf3355f05330ff5
r    =   48'hba2b754bd72d
```

3-rd round

- round

```
xl   =   32'h72731955
xr   =   32'hf4bb5951
k    =   48'h8d762d5a7da8
rl   =   32'hf4bb5951
rr   =   32'h022d760b
```

- perm_E

```
x    =   32'hf4bb5951
r    =   48'hfa95f6af2aa3
```

- split_1

```
x    =   48'h77e3dbf5570b
r0   =   6'h0b
r1   =   6'h1c
r2   =   6'h15
r3   =   6'h3d
r4   =   6'h1b
r5   =   6'h0f
r6   =   6'h3e
r7   =   6'h1d
```

- sbox_0

```
x    =  6'h0b
r    =  4'h3
```

- sbox_1

```
x    =  6'h1c
r    =  4'h6
```

- sbox_2

```
x    =  6'h15
r    =  4'hd
```

- sbox_3

```
x    =  6'h3d
r    =  4'h5
```

- sbox_4

```
x    =  6'h1b
r    =  4'ha
```

- sbox_5

```
x    =  6'h0f
r    =  4'ha
```

- sbox_6

```
x    =  6'h3e
r    =  4'hf
```

- sbox_7

```
x    =  6'h1d
r    =  4'h3
```

- merge_1

```
x0   =  6'h3
x1   =  6'h6
x2   =  6'hd
x3   =  6'h5
x4   =  6'ha
x5   =  6'ha
x6   =  6'hf
x7   =  6'h3
r    =  48'h3faa5d63
```

- perm_P

```
x    =  32'h3faa5d63
r    =  32'h705e6f5e
```

- key_schedule

```
x    =  56'hf3355f05330ff5
i    =  4'h3
r    =  56'hccd57c34cc3fd5
k    =  48'h8d762d5a7da8
```

  * split_0

```
x    =  56'hf3355f05330ff5
r0   =  28'h5330ff5
r1   =  28'hf3355f0
```

  * clr_28bit (left-hand instance)

```
x    =  28'hf3355f0
y    =  4'h3
r    =  28'hccd57c3
```

  * clr_28bit (right-hand instance)

```
x    =  28'h5330ff5
y    =  4'h3
r    =  28'h4cc3fd5
```

  * merge_0

```
x0   =  28'h4cc3fd5
x1   =  28'hccd57c3
r    =  56'hccd57c34cc3fd5
```

  * perm_PC2

```
x    =  56'hccd57c34cc3fd5
r    =  48'h8d762d5a7da8
```

4-th round

- round

```
xl   =  32'hf4bb5951
xr   =  32'h022d760b
k    =  48'hc317fce8593d
rl   =  32'h022d760b
rr   =  32'h8ea83bc2
```

  - perm_E

```
x    =  32'h022d760b
r    =  48'h80415abac056
```

  - split_1

```
x    =  48'h4356a652996b
r0   =  6'h2b
r1   =  6'h25
r2   =  6'h29
r3   =  6'h14
r4   =  6'h26
r5   =  6'h1a
r6   =  6'h35
r7   =  6'h10
```

  - sbox_0

```
x    =  6'h2b
r    =  4'ha
```

  - sbox_1

```
x    =  6'h25
r    =  4'hd
```

  - sbox_2

```
x    =  6'h29
r    =  4'h9
```

  - sbox_3

```
x    =  6'h14
r    =  4'h3
```

  - sbox_4

```
x    =  6'h26
r    =  4'h0
```

  - sbox_5

```
x    =  6'h1a
r    =  4'h4
```

  - sbox_6

```
x    =  6'h35
r    =  4'h7
```

  - sbox_7

```
x    =  6'h10
r    =  4'h3
```

  - merge_1

```
x0   =  6'ha
x1   =  6'hd
x2   =  6'h9
x3   =  6'h3
x4   =  6'h0
x5   =  6'h4
x6   =  6'h7
x7   =  6'h3
r    =  48'h374039da
```

  - perm_P

```
x    =  32'h374039da
r    =  32'h7a136293
```

  - key_schedule

```
x    =  56'hccd57c34cc3fd5
i    =  4'h4
r    =  56'h3355f0f330ff55
k    =  48'hc317fce8593d
```

* split_0
```
x    =  56'hccd57c34cc3fd5
r0   =  28'h4cc3fd5
r1   =  28'hccd57c3
```
* clr_28bit (left-hand instance)
```
x    =  28'hccd57c3
y    =  4'h4
r    =  28'h3355f0f
```
* clr_28bit (right-hand instance)
```
x    =  28'h4cc3fd5
y    =  4'h4
r    =  28'h330ff55
```
* merge_0
```
x0   =  28'h330ff55
x1   =  28'h3355f0f
r    =  56'h3355f0f330ff55
```
* perm_PC2
```
x    =  56'h3355f0f330ff55
r    =  48'hc317fce8593d
```

5-th round

- round
```
xl   =  32'h022d760b
xr   =  32'h8ea83bc2
k    =  48'hdcdae1c37aba
rl   =  32'h8ea83bc2
rr   =  32'hbfb63e46
```
  - perm_E
```
x    =  32'h8ea83bc2
r    =  48'h45d5501f7e05
```
  - split_1
```
x    =  48'h990fb1dc04bf
r0   =  6'h3f
r1   =  6'h12
r2   =  6'h00
r3   =  6'h37
r4   =  6'h31
r5   =  6'h3e
r6   =  6'h10
r7   =  6'h26
```
  - sbox_0
```
x    =  6'h3f
r    =  4'hb
```
  - sbox_1
```
x    =  6'h12
r    =  4'hc
```
  - sbox_2
```
x    =  6'h00
r    =  4'hc
```
  - sbox_3
```
x    =  6'h37
r    =  4'h9
```
  - sbox_4
```
x    =  6'h31
r    =  4'h9
```

- sbox_5
```
x    =  6'h3e
r    =  4'h7
```
- sbox_6
```
x    =  6'h10
r    =  4'h9
```
- sbox_7
```
x    =  6'h26
r    =  4'h8
```
- merge_1
```
x0   =  6'hb
x1   =  6'hc
x2   =  6'hc
x3   =  6'h9
x4   =  6'h9
x5   =  6'h7
x6   =  6'h9
x7   =  6'h8
r    =  48'h89799ccb
```
- perm_P
```
x    =  32'h89799ccb
r    =  32'hbd9b484d
```
- key_schedule
```
x    =  56'h3355f0f330ff55
i    =  4'h5
r    =  56'hcd57c3ccc3fd54
k    =  48'hdcdae1c37aba
```
  * split_0
```
x    =  56'h3355f0f330ff55
r0   =  28'h330ff55
r1   =  28'h3355f0f
```
  * clr_28bit (left-hand instance)
```
x    =  28'h3355f0f
y    =  4'h5
r    =  28'hcd57c3c
```
  * clr_28bit (right-hand instance)
```
x    =  28'h330ff55
y    =  4'h5
r    =  28'hcc3fd54
```
  * merge_0
```
x0   =  28'hcc3fd54
x1   =  28'hcd57c3c
r    =  56'hcd57c3ccc3fd54
```
  * perm_PC2
```
x    =  56'hcd57c3ccc3fd54
r    =  48'hdcdae1c37aba
```

6-th round

- round
```
xl   =  32'h8ea83bc2
xr   =  32'hbfb63e46
k    =  48'h93fb6af51b39
rl   =  32'hbfb63e46
rr   =  32'h70425c5d
```
- perm_E
```
x    =  32'hbfb63e46
r    =  48'h5ffdac1fc20d
```
- split_1

```
x    =  48'hcc06c6ead934
r0   =  6'h34
r1   =  6'h24
r2   =  6'h2d
r3   =  6'h3a
r4   =  6'h06
r5   =  6'h1b
r6   =  6'h00
r7   =  6'h33
```

- sbox_0

```
x    =  6'h34
r    =  4'ha
```

- sbox_1

```
x    =  6'h24
r    =  4'hb
```

- sbox_2

```
x    =  6'h2d
r    =  4'hf
```

- sbox_3

```
x    =  6'h3a
r    =  4'h3
```

- sbox_4

```
x    =  6'h06
r    =  4'h3
```

- sbox_5

```
x    =  6'h1b
r    =  4'hb
```

- sbox_6

```
x    =  6'h00
r    =  4'hf
```

- sbox_7

```
x    =  6'h33
r    =  4'hb
```

- merge_1

```
x0   =  6'ha
x1   =  6'hb
x2   =  6'hf
x3   =  6'h3
x4   =  6'h3
x5   =  6'hb
x6   =  6'hf
x7   =  6'hb
r    =  48'hbfb33fba
```

- perm_P

```
x    =  32'hbfb33fba
r    =  32'hfeea679f
```

- key_schedule

```
x    =  56'hcd57c3ccc3fd54
i    =  4'h6
r    =  56'h355f0f330ff553
k    =  48'h93fb6af51b39
```

  * split_0

```
x    =  56'hcd57c3ccc3fd54
r0   =  28'hcc3fd54
r1   =  28'hcd57c3c
```

  * clr_28bit (left-hand instance)

```
x    =  28'hcd57c3c
y    =  4'h6
r    =  28'h355f0f3
```

  * clr_28bit (right-hand instance)

```
x    =  28'hcc3fd54
y    =  4'h6
r    =  28'h30ff553
```

* merge_0

```
x0   =  28'h30ff553
x1   =  28'h355f0f3
r    =  56'h355f0f330ff553
```

* perm_PC2

```
x    =  56'h355f0f330ff553
r    =  48'h93fb6af51b39
```

7-th round

- round

```
xl   =  32'hbfb63e46
xr   =  32'h70425c5d
k    =  48'ha877c7931a7e
rl   =  32'h70425c5d
rr   =  32'h9603df08
```

  - perm_E

```
x    =  32'h70425c5d
r    =  48'hba02042f82fa
```

  - split_1

```
x    =  48'h1275c3bc9884
r0   =  6'h04
r1   =  6'h22
r2   =  6'h09
r3   =  6'h2f
r4   =  6'h03
r5   =  6'h17
r6   =  6'h27
r7   =  6'h04
```

  - sbox_0

```
x    =  6'h04
r    =  4'h8
```

  - sbox_1

```
x    =  6'h22
r    =  4'h4
```

  - sbox_2

```
x    =  6'h09
r    =  4'h7
```

  - sbox_3

```
x    =  6'h2f
r    =  4'hd
```

  - sbox_4

```
x    =  6'h03
r    =  4'h8
```

  - sbox_5

```
x    =  6'h17
r    =  4'he
```

  - sbox_6

```
x    =  6'h27
r    =  4'h1
```

  - sbox_7

```
x    =  6'h04
r    =  4'hd
```

  - merge_1

```
x0   =   6'h8
x1   =   6'h4
x2   =   6'h7
x3   =   6'hd
x4   =   6'h8
x5   =   6'he
x6   =   6'h1
x7   =   6'hd
r    =   48'hd1e8d748
```

- perm_P

```
x    =   32'hd1e8d748
r    =   32'h29b5e14e
```

- key_schedule

```
x    =   56'h355f0f330ff553
i    =   4'h7
r    =   56'hd57c3ccc3fd54c
k    =   48'ha877c7931a7e
```

  * split_0

```
x    =   56'h355f0f330ff553
r0   =   28'h30ff553
r1   =   28'h355f0f3
```

  * clr_28bit (left-hand instance)

```
x    =   28'h355f0f3
y    =   4'h7
r    =   28'hd57c3cc
```

  * clr_28bit (right-hand instance)

```
x    =   28'h30ff553
y    =   4'h7
r    =   28'hc3fd54c
```

  * merge_0

```
x0   =   28'hc3fd54c
x1   =   28'hd57c3cc
r    =   56'hd57c3ccc3fd54c
```

  * perm_PC2

```
x    =   56'hd57c3ccc3fd54c
r    =   48'ha877c7931a7e
```

8-th round

- round

```
xl   =   32'h70425c5d
xr   =   32'h9603df08
k    =   48'h3f3616d947c6
rl   =   32'h9603df08
rr   =   32'h4d44034b
```

- perm_E

```
x    =   32'h9603df08
r    =   48'h4ac007efe851
```

- split_1

```
x    =   48'h75f61136af97
r0   =   6'h17
r1   =   6'h3e
r2   =   6'h2a
r3   =   6'h0d
r4   =   6'h11
r5   =   6'h18
r6   =   6'h1f
r7   =   6'h1d
```

- sbox_0

```
x    =  6'h17
r    =  4'hb
```

- sbox_1

```
x    =  6'h3e
r    =  4'h2
```

- sbox_2

```
x    =  6'h2a
r    =  4'h8
```

- sbox_3

```
x    =  6'h0d
r    =  4'hd
```

- sbox_4

```
x    =  6'h11
r    =  4'h4
```

- sbox_5

```
x    =  6'h18
r    =  4'hb
```

- sbox_6

```
x    =  6'h1f
r    =  4'h5
```

- sbox_7

```
x    =  6'h1d
r    =  4'h3
```

- merge_1

```
x0   =  6'hb
x1   =  6'h2
x2   =  6'h8
x3   =  6'hd
x4   =  6'h4
x5   =  6'hb
x6   =  6'h5
x7   =  6'h3
r    =  48'h35b4d82b
```

- perm_P

```
x    =  32'h35b4d82b
r    =  32'h3d065f16
```

- key_schedule

```
x    =  56'hd57c3ccc3fd54c
i    =  4'h8
r    =  56'haaf879987faa99
k    =  48'h3f3616d947c6
```

  * split_0

```
x    =  56'hd57c3ccc3fd54c
r0   =  28'hc3fd54c
r1   =  28'hd57c3cc
```

  * clr_28bit (left-hand instance)

```
x    =  28'hd57c3cc
y    =  4'h8
r    =  28'haaf8799
```

  * clr_28bit (right-hand instance)

```
x    =  28'hc3fd54c
y    =  4'h8
r    =  28'h87faa99
```

  * merge_0

```
x0   =  28'h87faa99
x1   =  28'haaf8799
r    =  56'haaf879987faa99
```

  * perm_PC2

```
x    =  56'haaf879987faa99
r    =  48'h3f3616d947c6
```

9-th round

- round

```
xl   =  32'h9603df08
xr   =  32'h4d44034b
k    =  48'h6e1cf89ce28d
rl   =  32'h4d44034b
rr   =  32'h28c52afd
```

  - perm_E

```
x    =  32'h4d44034b
r    =  48'ha5aa08006a56
```

  - split_1

```
x    =  48'hcbb6f09c88db
r0   =  6'h1b
r1   =  6'h23
r2   =  6'h08
r3   =  6'h27
r4   =  6'h30
r5   =  6'h1b
r6   =  6'h3b
r7   =  6'h32
```

  - sbox_0

```
x    =  6'h1b
r    =  4'he
```

  - sbox_1

```
x    =  6'h23
r    =  4'hb
```

  - sbox_2

```
x    =  6'h08
r    =  4'h9
```

  - sbox_3

```
x    =  6'h27
r    =  4'h7
```

  - sbox_4

```
x    =  6'h30
r    =  4'hf
```

  - sbox_5

```
x    =  6'h1b
r    =  4'hb
```

  - sbox_6

```
x    =  6'h3b
r    =  4'h5
```

  - sbox_7

```
x    =  6'h32
r    =  4'hc
```

  - merge_1

```
x0   =  6'he
x1   =  6'hb
x2   =  6'h9
x3   =  6'h7
x4   =  6'hf
x5   =  6'hb
x6   =  6'h5
x7   =  6'hc
r    =  48'hc5bf79be
```

  - perm_P

```
x    =  32'hc5bf79be
r    =  32'hbec6f5f5
```

  - key_schedule

```
x    =  56'haaf879987faa99
i    =  4'h9
r    =  56'habe1e661feaa66
k    =  48'h6e1cf89ce28d
```

* split_0
```
x    =  56'haaf879987faa99
r0   =  28'h87faa99
r1   =  28'haaf8799
```
* clr_28bit (left-hand instance)
```
x    =  28'haaf8799
y    =  4'h9
r    =  28'habe1e66
```
* clr_28bit (right-hand instance)
```
x    =  28'h87faa99
y    =  4'h9
r    =  28'h1feaa66
```
* merge_0
```
x0   =  28'h1feaa66
x1   =  28'habe1e66
r    =  56'habe1e661feaa66
```
* perm_PC2
```
x    =  56'habe1e661feaa66
r    =  48'h6e1cf89ce28d
```

10-th round

- round
```
xl   =  32'h4d44034b
xr   =  32'h28c52afd
k    =  48'hdee07cf276c5
rl   =  32'h28c52afd
rr   =  32'h658f5c8b
```
  - perm_E
```
x    =  32'h28c52afd
r    =  48'h95160a9557fa
```
  - split_1
```
x    =  48'h4bf67667213f
r0   =  6'h3f
r1   =  6'h04
r2   =  6'h32
r3   =  6'h19
r4   =  6'h36
r5   =  6'h19
r6   =  6'h3f
r7   =  6'h12
```
  - sbox_0
```
x    =  6'h3f
r    =  4'hb
```
  - sbox_1
```
x    =  6'h04
r    =  4'h2
```
  - sbox_2
```
x    =  6'h32
r    =  4'h0
```
  - sbox_3
```
x    =  6'h19
r    =  4'h3
```
  - sbox_4
```
x    =  6'h36
r    =  4'he
```

- sbox_5
```
x    =  6'h19
r    =  4'hc
```
- sbox_6
```
x    =  6'h3f
r    =  4'h9
```
- sbox_7
```
x    =  6'h12
r    =  4'ha
```
- merge_1
```
x0   =  6'hb
x1   =  6'h2
x2   =  6'h0
x3   =  6'h3
x4   =  6'he
x5   =  6'hc
x6   =  6'h9
x7   =  6'ha
r    =  48'ha9ce302b
```
- perm_P
```
x    =  32'ha9ce302b
r    =  32'h28cb5fc0
```
- key_schedule
```
x    =  56'habe1e661feaa66
i    =  4'ha
r    =  56'haf8799a7faa998
k    =  48'hdee07cf276c5
```
  * split_0
```
x    =  56'habe1e661feaa66
r0   =  28'h1feaa66
r1   =  28'habe1e66
```
  * clr_28bit (left-hand instance)
```
x    =  28'habe1e66
y    =  4'ha
r    =  28'haf8799a
```
  * clr_28bit (right-hand instance)
```
x    =  28'h1feaa66
y    =  4'ha
r    =  28'h7faa998
```
  * merge_0
```
x0   =  28'h7faa998
x1   =  28'haf8799a
r    =  56'haf8799a7faa998
```
  * perm_PC2
```
x    =  56'haf8799a7faa998
r    =  48'hdee07cf276c5
```

11-st round

- round
```
xl   =  32'h28c52afd
xr   =  32'h658f5c8b
k    =  48'h8ecf1abaa3ab
rl   =  32'h658f5c8b
rr   =  32'h28965e1b
```
- perm_E
```
x    =  32'h658f5c8b
r    =  48'hb0bc5eaf9456
```
- split_1

```
x    =  48'h3e73441537fd
r0   =  6'h3d
r1   =  6'h1f
r2   =  6'h13
r3   =  6'h05
r4   =  6'h04
r5   =  6'h0d
r6   =  6'h27
r7   =  6'h0f
```

- sbox_0

```
x    =  6'h3d
r    =  4'h6
```

- sbox_1

```
x    =  6'h1f
r    =  4'h6
```

- sbox_2

```
x    =  6'h13
r    =  4'h1
```

- sbox_3

```
x    =  6'h05
r    =  4'h2
```

- sbox_4

```
x    =  6'h04
r    =  4'he
```

- sbox_5

```
x    =  6'h0d
r    =  4'h6
```

- sbox_6

```
x    =  6'h27
r    =  4'h1
```

- sbox_7

```
x    =  6'h0f
r    =  4'h1
```

- merge_1

```
x0   =  6'h6
x1   =  6'h6
x2   =  6'h1
x3   =  6'h2
x4   =  6'he
x5   =  6'h6
x6   =  6'h1
x7   =  6'h1
r    =  48'h116e2166
```

- perm_P

```
x    =  32'h116e2166
r    =  32'h005374e6
```

- key_schedule

```
x    =  56'haf8799a7faa998
i    =  4'hb
r    =  56'hbe1e66afeaa661
k    =  48'h8ecf1abaa3ab
```

  * split_0

```
x    =  56'haf8799a7faa998
r0   =  28'h7faa998
r1   =  28'haf8799a
```

  * clr_28bit (left-hand instance)

```
x    =  28'haf8799a
y    =  4'hb
r    =  28'hbe1e66a
```

  * clr_28bit (right-hand instance)

```
x    =  28'h7faa998
y    =  4'hb
r    =  28'hfeaa661
```

* merge_0

```
x0   =  28'hfeaa661
x1   =  28'hbe1e66a
r    =  56'hbe1e66afeaa661
```

* perm_PC2

```
x    =  56'hbe1e66afeaa661
r    =  48'h8ecf1abaa3ab
```

12-nd round

- round

```
xl   =  32'h658f5c8b
xr   =  32'h28965e1b
k    =  48'h6e3b2fb67f03
rl   =  32'h28965e1b
rr   =  32'h58b6744e
```

  - perm_E

```
x    =  32'h28965e1b
r    =  48'h9514ac2fc0f6
```

  - split_1

```
x    =  48'hfb2f8399bff5
r0   =  6'h35
r1   =  6'h3f
r2   =  6'h1b
r3   =  6'h26
r4   =  6'h03
r5   =  6'h3e
r6   =  6'h32
r7   =  6'h3e
```

  - sbox_0

```
x    =  6'h35
r    =  4'h9
```

  - sbox_1

```
x    =  6'h3f
r    =  4'hc
```

  - sbox_2

```
x    =  6'h1b
r    =  4'hb
```

  - sbox_3

```
x    =  6'h26
r    =  4'hb
```

  - sbox_4

```
x    =  6'h03
r    =  4'h8
```

  - sbox_5

```
x    =  6'h3e
r    =  4'h7
```

  - sbox_6

```
x    =  6'h32
r    =  4'h8
```

  - sbox_7

```
x    =  6'h3e
r    =  4'h0
```

  - merge_1

```
x0   =  6'h9
x1   =  6'hc
x2   =  6'hb
x3   =  6'hb
x4   =  6'h8
x5   =  6'h7
x6   =  6'h8
x7   =  6'h0
r    =  48'h0878bbc9
```

- perm_P

```
x    =  32'h0878bbc9
r    =  32'h3d3928c5
```

- key_schedule

```
x    =  56'hbe1e66afeaa661
i    =  4'hc
r    =  56'hf8799aafaa9987
k    =  48'h6e3b2fb67f03
```

  * split_0

```
x    =  56'hbe1e66afeaa661
r0   =  28'hfeaa661
r1   =  28'hbe1e66a
```

  * clr_28bit (left-hand instance)

```
x    =  28'hbe1e66a
y    =  4'hc
r    =  28'hf8799aa
```

  * clr_28bit (right-hand instance)

```
x    =  28'hfeaa661
y    =  4'hc
r    =  28'hfaa9987
```

  * merge_0

```
x0   =  28'hfaa9987
x1   =  28'hf8799aa
r    =  56'hf8799aafaa9987
```

  * perm_PC2

```
x    =  56'hf8799aafaa9987
r    =  48'h6e3b2fb67f03
```

13-rd round

- round

```
xl   =  32'h28965e1b
xr   =  32'h58b6744e
k    =  48'habbc497e2372
rl   =  32'h58b6744e
rr   =  32'h48389819
```

- perm_E

```
x    =  32'h58b6744e
r    =  48'h2f15ac3a825c
```

- split_1

```
x    =  48'h84a9e544a12e
r0   =  6'h2e
r1   =  6'h04
r2   =  6'h0a
r3   =  6'h11
r4   =  6'h25
r5   =  6'h27
r6   =  6'h0a
r7   =  6'h21
```

- sbox_0

```
x    =  6'h2e
r    =  4'h2
```

- sbox_1

```
x    =  6'h04
r    =  4'h2
```

- sbox_2

```
x    =  6'h0a
r    =  4'h2
```

- sbox_3

```
x    =  6'h11
r    =  4'h5
```

- sbox_4

```
x    =  6'h25
r    =  4'h0
```

- sbox_5

```
x    =  6'h27
r    =  4'h0
```

- sbox_6

```
x    =  6'h0a
r    =  4'hb
```

- sbox_7

```
x    =  6'h21
r    =  4'hf
```

- merge_1

```
x0   =  6'h2
x1   =  6'h2
x2   =  6'h2
x3   =  6'h5
x4   =  6'h0
x5   =  6'h0
x6   =  6'hb
x7   =  6'hf
r    =  48'hfb005222
```

- perm_P

```
x    =  32'hfb005222
r    =  32'h60aec602
```

- key_schedule

```
x    =  56'hf8799aafaa9987
i    =  4'hd
r    =  56'he1e66abeaa661f
k    =  48'habbc497e2372
```

  * split_0

```
x    =  56'hf8799aafaa9987
r0   =  28'hfaa9987
r1   =  28'hf8799aa
```

  * clr_28bit (left-hand instance)

```
x    =  28'hf8799aa
y    =  4'hd
r    =  28'he1e66ab
```

  * clr_28bit (right-hand instance)

```
x    =  28'hfaa9987
y    =  4'hd
r    =  28'heaa661f
```

  * merge_0

```
x0   =  28'heaa661f
x1   =  28'he1e66ab
r    =  56'he1e66abeaa661f
```

  * perm_PC2

```
x    =  56'he1e66abeaa661f
r    =  48'habbc497e2372
```

14-th round

- round

```
xl   =  32'h58b6744e
xr   =  32'h48389819
k    =  48'h496efaf5e94a
rl   =  32'h48389819
rr   =  32'h93cd4d3b
```

  - perm_E

```
x    =  32'h48389819
r    =  48'ha501f14f00f2
```

  - split_1

```
x    =  48'hec6f0bbae9b8
r0   =  6'h38
r1   =  6'h26
r2   =  6'h2e
r3   =  6'h2e
r4   =  6'h0b
r5   =  6'h3c
r6   =  6'h06
r7   =  6'h3b
```

  - sbox_0

```
x    =  6'h38
r    =  4'hf
```

  - sbox_1

```
x    =  6'h26
r    =  4'hd
```

  - sbox_2

```
x    =  6'h2e
r    =  4'h3
```

  - sbox_3

```
x    =  6'h2e
r    =  4'h8
```

  - sbox_4

```
x    =  6'h0b
r    =  4'hf
```

  - sbox_5

```
x    =  6'h3c
r    =  4'he
```

  - sbox_6

```
x    =  6'h06
r    =  4'he
```

  - sbox_7

```
x    =  6'h3b
r    =  4'h0
```

  - merge_1

```
x0   =  6'hf
x1   =  6'hd
x2   =  6'h3
x3   =  6'h8
x4   =  6'hf
x5   =  6'he
x6   =  6'he
x7   =  6'h0
r    =  48'h0eef83df
```

  - perm_P

```
x    =  32'h0eef83df
r    =  32'hcb7b3975
```

  - key_schedule

```
x    =  56'he1e66abeaa661f
i    =  4'he
r    =  56'h8799aafaa9987f
k    =  48'h496efaf5e94a
```

* split_0
```
x    =  56'he1e66abeaa661f
r0   =  28'heaa661f
r1   =  28'he1e66ab
```
* clr_28bit (left-hand instance)
```
x    =  28'he1e66ab
y    =  4'he
r    =  28'h8799aaf
```
* clr_28bit (right-hand instance)
```
x    =  28'heaa661f
y    =  4'he
r    =  28'haa9987f
```
* merge_0
```
x0   =  28'haa9987f
x1   =  28'h8799aaf
r    =  56'h8799aafaa9987f
```
* perm_PC2
```
x    =  56'h8799aafaa9987f
r    =  48'h496efaf5e94a
```

15-th round

- round
```
xl   =  32'h48389819
xr   =  32'h93cd4d3b
k    =  48'h35c2fc478fcd
rl   =  32'h93cd4d3b
rr   =  32'h5e8e5083
```
  - perm_E
```
x    =  32'h93cd4d3b
r    =  48'hca7e5aa5a9f7
```
  - split_1
```
x    =  48'hffbca6e2263a
r0   =  6'h3a
r1   =  6'h18
r2   =  6'h22
r3   =  6'h38
r4   =  6'h26
r5   =  6'h32
r6   =  6'h3b
r7   =  6'h3f
```
  - sbox_0
```
x    =  6'h3a
r    =  4'h3
```
  - sbox_1
```
x    =  6'h18
r    =  4'h5
```
  - sbox_2
```
x    =  6'h22
r    =  4'he
```
  - sbox_3
```
x    =  6'h38
r    =  4'h6
```
  - sbox_4
```
x    =  6'h26
r    =  4'h0
```

- sbox_5
```
x    =  6'h32
r    =  4'h1
```
- sbox_6
```
x    =  6'h3b
r    =  4'h5
```
- sbox_7
```
x    =  6'h3f
r    =  4'hd
```
- merge_1
```
x0   =  6'h3
x1   =  6'h5
x2   =  6'he
x3   =  6'h6
x4   =  6'h0
x5   =  6'h1
x6   =  6'h5
x7   =  6'hd
r    =  48'hd5106e53
```
- perm_P
```
x    =  32'hd5106e53
r    =  32'h16b6c89a
```
- key_schedule
```
x    =  56'h8799aafaa9987f
i    =  4'hf
r    =  56'h0f3355f55330ff
k    =  48'h35c2fc478fcd
```
  * split_0
```
x    =  56'h8799aafaa9987f
r0   =  28'haa9987f
r1   =  28'h8799aaf
```
  * clr_28bit (left-hand instance)
```
x    =  28'h8799aaf
y    =  4'hf
r    =  28'h0f3355f
```
  * clr_28bit (right-hand instance)
```
x    =  28'haa9987f
y    =  4'hf
r    =  28'h55330ff
```
  * merge_0
```
x0   =  28'h55330ff
x1   =  28'h0f3355f
r    =  56'h0f3355f55330ff
```
  * perm_PC2
```
x    =  56'h0f3355f55330ff
r    =  48'h35c2fc478fcd
```

Post-processing

- merge_2
```
x0   =  32'h93cd4d3b
x1   =  32'h5e8e5083
r    =  64'h5e8e508393cd4d3b
```
- perm_FP
```
x    =  64'h5e8e508393cd4d3b
r    =  64'habd3787ac6026cb1
```

Output

```
c    =  64'habd3787ac6026cb1
```