1 MRSW registers from SRSW registers

In this section we shall see how to build Multiple Reader Single Writer registers from Single Reader Single Writer registers. When a MRSW register is simulated from SRSW registers, we call the MRSW register the logical register or high level register and the SRSW registers as low level registers or physical registers. Physical registers are the building blocks for logical registers. Note that all logical registers and physical registers discussed in this section are wait-free. That is, if there is an action by user to the register, the user will receive a response even of other users are accessing the same register.

The following figure shows how we can simulate MRSW register from SRSW register.

The registers $X_1, ..., X_n$ are the SRSW physical registers. The bigger rectangle represents the MRSW logical register. In the rest of this document read and write operations on logical registers are denoted by upper case alphabet (READ and WRITE) and read and write on physical registers are written in lower case. Every operation has an invocation and a response. The following algorithms for READ and WRITE can be used to implement MRSW register from SRSW registers.
WRITE(v):
for all i in {1,...,n}
do
    write(v) on X_i
wait for ack_i from all X_i
ACK

READ_i:
read(X_i)
wait for return_i(v)
RETURN_i(v)

Lemma 1: If $X_1,...,X_n$ are safe registers then so is the logical register implemented by the above READ and WRITE operations.

Proof: On read operation a safe register has to return the value of latest completed write when there are no overlapping writes. Consider any $READ_i$ of MRSW register with no overlapping WRITEs. If the logical READ does not overlap with any logical WRITEs then the physical read which is part of the logical READ will also not overlap with any logical WRITEs. That means physical reads will not overlap with any of the physical writes. Under such conditions, SRSW safe registers return value of latest completed write. So, the $READ_i$ operation in this case will also return the value of latest completed logical WRITE. So, the logical register is safe.

Lemma 2: If $X_1,...,X_n$ are regular registers then so is the above described logical register.

Proof: Following the same argument as previous proof the statement of the lemma holds for READs with no overlapping WRITEs. Consider the case where the READ overlaps with some non zero number of WRITEs. As each of the $X_i$ is a regular register, it will return either the value of most recent write or the value of the overlapping write. Any logical $READ_i$ reads only one physical register. When the $READ_i$ overlaps with a WRITE operation, the physical read within $READ_i$ overlaps with the WRITE operation.

From the point of view of $i$th register one of the following 3 scenarios will happen:
1. The physical read can happen before the physical write on $X_i$ happens. In this case, the READ returns the value of last completed WRITE (we can be sure that the WRITE is a “completed” WRITE as there is a single writer).
2. The physical read overlaps with the physical write on $X_i$. Then the register returns either the previous value of $X_i$ or the currently updated value of $X_i$ because $X_i$ is a regular register. In this case also, given the algorithm, it will return the value of previous WRITE or the overlapping WRITE.
3. The physical read happens after the physical write on $X_i$. In this case $READ_i$ returns the updated value of $X_i$ which is also the value being written to the logical register.

So, in all three cases, the logical READ returns either value of the latest completed WRITE or the overlapping WRITE.
Using the above ideas we can construct $k$-ary registers from binary registers. Suppose $k = \lfloor 2^l \rfloor$ then we need at least $l$ binary registers to implement a $k$-ary register. The following algorithms can be used for reading and writing the logical register:

\[ \text{WRITE}(v): \]
for $i$ in \{1,...,l\}
write bit $i$ of $v$ into $X_i$
wait for ack

\[ \text{READ}: \]
for $i$ in \{1,...,l\}
read bit $i$ of $v$ from $X_i$

\[ \text{ACK} \]

\[ \text{RETURN}(v) \]

\[ \text{ACK} \]

**Lemma 3:** If $X_1,...,X_l$ are binary safe, then the $k$-ary register implemented is also safe.

**Proof:** The requirement for a safe register is that it should return the value of the last completed WRITE when there are no overlapping WRITEs during the READ. In case of the above register, suppose a READ happens and no WRITE overlaps it. Suppose the value written by the latest WRITE be $v$. Then each bit of $v$ that is read is written by the last completed WRITE. As each $X_i$ is safe, the value of the logical register reflects the value written by the last completed WRITE. So, the $k$-ary register is safe.

Note that the above $k$-ary register is not regular even when each of the $X_i$ is regular. This can be proved using a counter example. Let the register be 8-ary. Then it will require 3 bits. Suppose the initial value or the value of latest WRITE be 0. A READ has commenced and read the least significant bit as 0. Before it reads the other 2 bits, writer has written the value 7 to the register. So it reads the other 2 bits to be 1 and returns the binary 110 (decimal: 6) as the value. However, 6 is neither the previous value nor the value written by the overlapping write. So, such implementation does not give a regular logical register.

In the following section, we shall modify the implementation of $k$-ary registers so that they can be regular.

## 2 Regular $k$-ary logical registers from regular binary registers

Consider the following algorithm for Writing and Reading $k$-ary registers (this implementation needs $k$ bits for implementing a $k$-ary register):

\[ \text{WRITE}(v): \]
write $X_v = 1$ for $i = 0$ to k-1
for $i = v-1$ to 0
write $X_i = 0$
ACK

\[ \text{READ}: \]
for $i = 0$ to k-1
read($X_i$) and wait for return
if $return_i(1)$ then RETURN (i)
If the physical registers \(X_1, \ldots, X_{k-1}\) are binary regular then the logical register implemented using above algorithms is \(k\)-ary regular. With out loss of generality suppose the register is initialized with a value of 0. Suppose no WRITE has taken place. Then the READ reads physical registers that has never been written before. In this case it returns the initial value.

In the other case the READ reads physical registers that have been written to by physical writes that are part of logical WRITES. In this case, the READ returns the value written by the **latest logical WRITE whose physical writes it reads**. To prove this statement formally we need the following definition and lemma.

**Definition 1** A logical READ, \(R\), reads from a logical WRITE, \(W\), if \(R\) contains a physical read that reads a physical write contained in \(W\).

**Lemma 4**: Let \(W\) be the latest WRITE which contains a physical write that is read by a physical read of READ, \(R\). Then \(R\) returns the logical value written by \(W\).

**Proof**: The trivial case is where no logical WRITE overlaps with the logical READ. In this case, the latest completed WRITE becomes the latest WRITE from which the reader reads. In this case the READ returns the value of latest WRITE and so, it is able to return the value of latest WRITE it reads from.

Suppose one or more than one WRITE overlaps with the READ operation. Consider a READ, \(R\), that is so slow that it reads from multiple logical WRITES (overlaps more than one WRITE).

Let \(W = WRITE(v)\) be the latest logical write that \(R\) reads from. Then, \(R\) reads \(X_v\) after \(W\) writes \(X_v = 1\) (because setting appropriate register to 1 is the first step of any WRITE operation). When \(R\) reads \(X_v\) it will read \(X_v = 1\) and returns \(v\). If it reads \(X_v = 0\) then it means, another WRITE, \(W_1\), has set \(X_v\) to 0 after \(W\) set \(X_v\) to 1. We can be sure that \(W_1\) comes after \(W\) because \(X_v = 0\) can only be set by a WRITE later than \(W\) (as WRITEs do not overlap). So, in this case, the physical read of \(X_v = 0\) is associated with \(W_1\) instead of \(W\). This contradicts the fact that \(W\) is the latest WRITE that \(R\) reads from. So, \(R\) can not read \(X_v\) to be 0. So, if \(W\) is the latest WRITE that \(R\) reads from then it is sure to read \(X_v\) as 1. So, it is sure to return \(v\).

Also, \(R\) can not read a 1 before \(v\). Let \(R\) read \(X_w\) for some \(w < v\) after \(W\) set \(X_w = 0\). Let the physical read of \(X_w\) by \(R\) happen at time \(t\). Since \(W\) is the latest WRITE that \(R\) reads from, \(R\) will read \(X_w = 0\) because no write can tamper with \(X_w\) before \(R\) reads it. If it did, then \(W\) will no longer be the latest WRITE that \(R\) reads from. For all registers \(X_{w+1}\) to \(X_{v-1}\), \(R\) will read 0 because they are written before time \(t\) and each of them changes only after \(R\) completes reading it because we know that \(W\) is the latest WRITE that \(R\) reads from. The same is the case with \(X_v\) - it is also set before \(t\) and does not change until \(R\) finishes reading it. So, \(R\) is sure to return \(v\) where \(W = WRITE(v)\).

**Theorem 1** Every logical read returns the value of the last preceding write or an overlapping write. If no preceding write exists it returns the initial value.
Proof: When there is no preceding write, the reader will read from only unwritten registers. Note that for the initial value to be an actual value in the current sense, at least one of the registers has to be 1. So, when the reader reads from unwritten registers, it is sure to read a 1 at some point and returns the appropriate value. The above lemma states that the READ always returns the value of the latest WRITE it reads from. We now claim that the latest WRITE that the reader reads from is either the last preceding write or an overlapping write. Suppose, the reader READs from an earlier WRITE, W. During the READ, when $X_0$ is read it returns the value written in $X_0$ by the latest preceding WRITE. But the latest preceding WRITE, $W'$, comes after $W$ according to our assumption. However, if READ can read a write by $W'$ at $X_0$ it self, there is no chance of it reading a write by $W$ at $X_i$ for $i > 0$. If it can, then it contradicts the fact that $W$ comes before $W'$ which results in the conclusion that $W$ can not be an earlier WRITE. So READ can only read from latest preceding WRITE or overlapping WRITEs. So, the READ returns either the value of an overlapping WRITE or the last preceding WRITE.