Contents

[1.Introduction 1](#_Toc1726475487)

[2.Methodology 2](#_Toc1493209814)

[2.1: APB Master 4](#_Toc233996949)

[2.2 APB Slave 4](#_Toc79311124)

[2.3. APB Operating States 5](#_Toc63538626)

[2.4:Write Transfer 6](#_Toc328580380)

[2.4.1: write with no wait Transfer: 6](#_Toc704586387)

[2.4.2: Write with no wait Transfer: 7](#_Toc1768882024)

[2.5: Read Transfer 7](#_Toc185856634)

[2.5.1: With no wait states 8](#_Toc338263293)

[2.5.2: With wait states 8](#_Toc580282821)

[3. UVM (Universal Verification Methodology) 8](#_Toc263539656)

[3.1: TestBench Architecture Overview 10](#_Toc1880711586)

[4: Test 11](#_Toc1195263988)

[4.1: Base Test 11](#_Toc826225271)

[4.2: Sanity Simple Test 12](#_Toc1454340273)

[4.2.1: Test Purpose 12](#_Toc1998175962)

[4.2.2: Sequences Used 12](#_Toc1017184312)

[4.2.3: Sequence Flow 12](#_Toc569019069)

[4.2.4: Additioal Arguments 13](#_Toc1328822303)

[4.3: Error Test 13](#_Toc2020543716)

[4.3.1: Test Purpose 13](#_Toc1497399545)

[4.3.2: Sequences used 13](#_Toc1687432662)

[4.3.3: Sequence Flow 14](#_Toc987234968)

[4.3.4: Additional Arguments 14](#_Toc1044521540)

[4.4: Continuous Write and Read Word Test(c\_wr\_rd\_word\_test) 14](#_Toc1123959408)

[4.4.1: Test Purpose 14](#_Toc1037706417)

[4.4.2: Sequences Used 14](#_Toc441566294)

[4.4.3: Sequence Flow 15](#_Toc2119418263)

[4.4.4: Additional Arguments 15](#_Toc676040480)

[4.5: Continuous write (Address increment), continuous read (Address increment): (c\_wr\_rd\_incr\_word\_test) 15](#_Toc108969430)

[4.5.1: Test Purpose 16](#_Toc1182924204)

[4.5.2: Sequences Used 16](#_Toc1287065052)

[4.5.3: Sequence Flow 16](#_Toc1212620403)

[4.5.4: Additional Arguments 16](#_Toc1367971439)

[4.6: Continuous write (Address increment), continuous read (Address decrement): (c\_wrIncr\_rdDecr\_word\_test) 17](#_Toc673726078)

[4.6.1: Test Purpose 17](#_Toc410151001)

[4.6.2: Sequences Used 17](#_Toc1202438850)

[4.6.3: Sequence Flow 18](#_Toc437372921)

[4.6.4: Additional Arguments 18](#_Toc1671207771)

[4.7: Continuous write (Address decrement), continuous read (Address increment): (c\_wrDecr\_rdIncr\_word\_test) 19](#_Toc1555677928)

[4.7.1: Test Purpose 19](#_Toc936307126)

[4.7.2: Sequences Used 19](#_Toc228039539)

[4.7.3: Sequence Flow 19](#_Toc1676631486)

[4.7.4: Additional Arguments 20](#_Toc106184585)

[4.8: Bulk write, bulk read(bulk\_wr\_rd\_word\_test) 20](#_Toc1947283427)

[4.8.1: Test Purpose 20](#_Toc1270103742)

[4.8.2: Sequences Used 20](#_Toc1228563407)

[4.8.3: Sequence Flow 21](#_Toc481111387)

[4.8.4: Additional Arguments 21](#_Toc2145378907)

[4.9: Continuous write (Address decrement), continuous read (Address decrement): (c\_wr\_rd\_decr\_word\_test) 22](#_Toc1858174910)

[4.9.1: Test Purpose 22](#_Toc1436966969)

[4.9.2: Sequences Used 22](#_Toc541818985)

[4.9.3: Sequence Flow 22](#_Toc878831336)

[4.9.4: Additional Arguments 23](#_Toc1467586651)

[4.10: Continuous write and read for both locations (1100, 0011): (c\_wr\_rd\_Hword\_BL\_test) 23](#_Toc1343155171)

[4.10.1: Test Purpose 24](#_Toc1110542583)

[4.10.2: Sequences Used 24](#_Toc1113977774)

[4.10.3: Sequence Flow 24](#_Toc1859755725)

[4.10.4: Additional Arguments 24](#_Toc204026270)

[4.11: Continuous write (Address increment), continuous read (Address increment) for both locations (1100, 0011): (c\_wr\_rd\_incr\_Hword\_BL\_test) 25](#_Toc1838706631)

[4.11.1: Test Purpose 25](#_Toc874914392)

[4.11.2: Sequences Used 25](#_Toc1583123459)

[4.11.3: Sequence Flow 26](#_Toc120376110)

[4.11.4: Additional Arguments 26](#_Toc214402097)

[4.12. Continuous write (Address increment), continuous read (Address decrement) for both locations (1100, 0011): c\_wrIncr\_rdDecr\_Hword\_BL\_test 26](#_Toc1007969818)

[4.12.1: Test Purpose 26](#_Toc382454141)

[4.12.2: Sequences Used 27](#_Toc1579662682)

[4.12.3: Sequence Flow 27](#_Toc5815650)

[4.12.4: Additional Arguments 27](#_Toc1932341361)

[4.13: Continuous write (Address decrement), continuous read (Address increment) for both locations (1100, 0011):( c\_wrDecr\_rdIncr\_Hword\_BL\_test) 28](#_Toc721737971)

[4.13.1: Test Purpose 28](#_Toc1954711081)

[4.13.2: Sequences Used 28](#_Toc2030719181)

[4.13.3: Sequence Flow 29](#_Toc1343652864)

[4.13.4: Additional Arguments 29](#_Toc1665675657)

[4.14: Bulk write, bulk read for both locations (1100, 0011): (bulk\_wr\_rd\_Hword\_BL\_test) 29](#_Toc586237050)

[4.14.1: Test Purpose 30](#_Toc1252094503)

[4.14.2: Sequences Used 30](#_Toc902952966)

[4.14.3: Sequence Flow 30](#_Toc1864692704)

[4.14.4: Additional Arguments 30](#_Toc2001098970)

[4.15: Continuous write (Address decrement), continuous read (Address decrement) for both locations (1100, 0011): (c\_wr\_rd\_decr\_Hword\_BL\_test) 31](#_Toc977573076)

[4.15.1: Test Purpose 31](#_Toc1509429554)

[4.15.2: Sequences Used 31](#_Toc1924833896)

[4.15.3: Sequence Flow 32](#_Toc840030700)

[4.15.4: Additional Arguments 32](#_Toc1957517150)

[4.16: Continuous write and read for location 1 (1100 or 0011): (c\_wr\_rd\_Hword\_L1\_test) 33](#_Toc269566786)

[4.16.1: Test Purpose 33](#_Toc534923116)

[4.16.2: Sequences Used 33](#_Toc1684604637)

[4.16.3: Sequence Flow 33](#_Toc191242018)

[4.16.4: Additional Arguments 34](#_Toc1027988674)

[4.17: Continuous write (Address increment), continuous read (Address increment) for location 1 (1100 or 0011): (c\_wr\_rd\_incr\_Hword\_L1\_test) 34](#_Toc1672447761)

[4.17.1: Test Purpose 34](#_Toc1170235004)

[4.17.2: Sequences Used 34](#_Toc1466876720)

[4.17.3: Sequence Flow 35](#_Toc2139235753)

[4.17.4: Additional Arguments 35](#_Toc14920020)

[4.18: Continuous write (Address increment), continuous read (Address decrement) for location 1 (1100 or 0011): (c\_wrIncr\_rdDecr\_Hword\_L1\_test.sv) 36](#_Toc2011001716)

[4.18.1: Test Purpose 36](#_Toc221042437)

[4.18.2: Sequences Used 36](#_Toc701988345)

[4.18.3: Sequence Flow 36](#_Toc197861631)

[4.18.4: Additional Arguments 37](#_Toc1469356756)

[4.19: Continuous write (Address decrement), continuous read (Address increment) for location 1 (1100 or 0011) : (c\_wrDecr\_rdIncr\_Hword\_L1\_test) 37](#_Toc1643958632)

[4.19.1: Test Purpose 38](#_Toc960103012)

[4.19.2: Sequences Used 38](#_Toc1331609598)

[4.19.3: Sequence Flow 38](#_Toc1756977312)

[4.19.4: Additional Arguments 38](#_Toc1352640393)

[4.20: Bulk write, bulk read for location 1 (1100 or 0011): (bulk\_wr\_rd\_Hword\_L1\_test) 39](#_Toc1221881558)

[4.20.1: Test Purpose 39](#_Toc639126901)

[4.20.2: Sequences Used 39](#_Toc1950388449)

[4.20.3: Sequence Flow 39](#_Toc2123588725)

[4.20.4: Additional Arguments 40](#_Toc491623204)

[4.21: Continuous write (Address decrement), continuous read (Address decrement) for location 1 (1100 or 0011): (c\_wr\_rd\_decr\_Hword\_L1\_test) 40](#_Toc1552522006)

[4.21.1: Test Purpose 40](#_Toc720911440)

[4.21.2: Sequences Used 40](#_Toc392169334)

[4.21.3: Sequence Flow 40](#_Toc914942937)

[4.21.4: Additional Arguments 41](#_Toc938089911)

[4.22: Bulk write, bulk read for both locations (1100, 0011) in random order : (bulk\_wr\_rd\_Hword\_BLRandSr\_test) 41](#_Toc873271741)

[4.22.1: Test Purpose 41](#_Toc1245362759)

[4.22.2: Sequences Used 41](#_Toc584939838)

[4.22.3: Sequence Flow 42](#_Toc162215552)

[4.22.4: Additional Arguments 42](#_Toc435027878)

[4.23: Continuous write (Address decrement), continuous read (Address decrement) for location 1 (1100 or 0011): (c\_wr\_rd\_decr\_Hword\_L1\_test) 43](#_Toc883693412)

[4.23.1: Test Purpose 43](#_Toc1649732402)

[4.23.2: Sequences Used 43](#_Toc1663582153)

[4.23.3: Sequence Flow 44](#_Toc1906819196)

[4.23.4: Additional Arguments 44](#_Toc1322821102)

[4.24: Continuous Write Read Byte All Location Increase Test(0001,0010,0100,1000): (c\_wr\_rd\_byte\_ALincr\_test) 44](#_Toc645227837)

[4.24.1: Test Purpose 45](#_Toc1265469252)

[4.24.2: Sequences Used 45](#_Toc1650460910)

[4.24.3: Sequence Flow 45](#_Toc1090041206)

[4.24.4: Additional Arguments 45](#_Toc1271628103)

[4.25: Continuous write (Address increment), continuous read (Address increment) for all byte locations (0001,0010,0100,100): (c\_wr\_rd\_incr\_byte\_ALIncr\_test) 46](#_Toc261524968)

[4.25.1: Test Purpose 46](#_Toc1132329558)

[4.25.2: Sequences Used 46](#_Toc539404237)

[4.25.3: Sequence Flow 47](#_Toc905489496)

[4.25.4: Additional Arguments 47](#_Toc1893497226)

[4.26. Continuous write (Address increment), continuous read (Address decrement) for both all byte locations (0001,0010,0100,1000): c\_wrIncr\_rdDecr\_byte\_ALIncr\_test 47](#_Toc1773746172)

[4.26.1: Test Purpose 48](#_Toc685718273)

[4.26.2: Sequences Used 48](#_Toc644277247)

[4.26.3: Sequence Flow 48](#_Toc1219188273)

[4.26.4: Additional Arguments 48](#_Toc521308744)

[4.27: Continuous write (Address decrement), continuous read (Address increment) for all byte locations (0001,0010,0100,1000): (c\_wrDecr\_rdIncr\_byte\_ALIncr\_test) 49](#_Toc228521195)

[4.27.1: Test Purpose 49](#_Toc1161853695)

[4.27.2: Sequences Used 49](#_Toc1499723834)

[4.27.3: Sequence Flow 50](#_Toc785530046)

[4.27.4: Additional Arguments 50](#_Toc149101693)

[4.28: Bulk write, bulk read for all byte locations (0001,0010,0100,1000): (bulk\_wr\_rd\_byte\_ALincr\_test) 51](#_Toc236651689)

[4.28.1: Test Purpose 51](#_Toc2063143023)

[4.28.2: Sequences Used 51](#_Toc1586008194)

[4.28.3: Sequence Flow 51](#_Toc135091241)

[4.28.4: Additional Arguments 52](#_Toc37307038)

[4.29: Continuous write (Address decrement), continuous read (Address decrement) for all byte locations (0001,0010,0100,1000): (c\_wr\_rd\_decr\_byte\_ALIncr\_test) 52](#_Toc1555666048)

[4.29.1: Test Purpose 52](#_Toc763289410)

[4.29.2: Sequences Used 53](#_Toc1468317392)

[4.29.3: Sequence Flow 53](#_Toc1722116029)

[4.29.4: Additional Arguments 53](#_Toc397683776)

[4.30: Continuous write and read for two random byte location RandL2(0010): (c\_wr\_rd\_byte\_RandL2\_test) 54](#_Toc962023067)

[4.30.1: Test Purpose 54](#_Toc1112749344)

[4.30.2: Sequences used 54](#_Toc1696305043)

[4.30.3: Sequence Flow 54](#_Toc1350087152)

[4.30.4: Additional Arguments 55](#_Toc1688912325)

[4.31: Continuous write (Address increment), continuous read (Address increment) for two random byte location: (c\_wr\_rd\_incr\_byte\_RandL2\_test) 55](#_Toc1387174704)

[4.31.1: Test Purpose 55](#_Toc2039298052)

[4.31.2: Sequences used 56](#_Toc1264375449)

[4.31.3: Sequence Flow 56](#_Toc412371322)

[4.31.4: Additional Arguments 56](#_Toc603224996)

[4.32: Continuous write (Address increment), continuous read (Address decrement) for two Random byte location: (c\_wrIncr\_rdDecr\_byte\_RandL2\_test) 57](#_Toc437063078)

[4.32.1: Test Purpose 57](#_Toc1179611715)

[4.32.2: Sequences used 58](#_Toc736253768)

[4.32.3: Sequence Flow 58](#_Toc29595191)

[4.32.4: Additional Arguments 58](#_Toc88366670)

[4.33: - Continuous write (Address decrement), continuous read (Address increment) for two random byte location: (c\_wrDecr\_rdIncr\_byte\_RandL2\_test) 59](#_Toc415274797)

[4.33.1: Test Purpose 59](#_Toc1061117422)

[4.33.2: Sequences used 59](#_Toc266385889)

[4.33.3: Sequence Flow 60](#_Toc306750742)

[4.33.4: Additional Arguments 60](#_Toc1697435175)

[4.34: Bulk write, bulk read for two random byte locations: (bulk wr\_rd\_byte\_RandL2\_test) 60](#_Toc1893738458)

[4.34.1: Test Purpose 61](#_Toc1823398761)

[4.34.2: Sequences used 61](#_Toc1050230263)

[4.34.4: Sequence Flow 61](#_Toc1195624040)

[4.34.4: Additional Arguments 61](#_Toc1036812332)

[4.35: Continuous write (Address decrement), continuous read (Address decrement) for two random byte locations: (c\_wr\_rd\_decr\_byte\_RandL2\_test) 62](#_Toc1951706202)

[4.35.1: Test Purpose 62](#_Toc76094480)

[4.35.2: Sequences used 62](#_Toc302336287)

[4.35.3: Sequence Flow 63](#_Toc1690872287)

[4.35.4: Additional Arguments 63](#_Toc301841328)

[4.36: Continuous write and read for random three byte locations: (c\_wr\_rd\_byte\_RandL3\_test) 63](#_Toc665398128)

[4.36.1: Test Purpose 64](#_Toc2085876366)

[4.36.2: Sequences used 64](#_Toc1733521176)

[4.36.3: Sequence Flow 64](#_Toc1108876890)

[4.36.4: Additional Arguments 64](#_Toc827175498)

[4.37: Continuous write (Address increment), continuous read (Address increment) for random three byte locations(c\_wr\_rd\_incr\_byte\_RandL3\_test) 65](#_Toc490285993)

[4.37.1: Test Purpose 65](#_Toc988887675)

[4.37.2: Sequences used 65](#_Toc1415886947)

[4.37.3: Sequence Flow 66](#_Toc1504649394)

[4.37.4: Additional Arguments 66](#_Toc913073675)

[4.38: Continuous write (Address increment), continuous read (Address decrement) for random three byte locations: (c\_wrIncr\_rdDecr\_byte\_RandL3\_test) 67](#_Toc319994208)

[4.38.1: Test Purpose 67](#_Toc1531740934)

[4.38.2: Sequences used 67](#_Toc389725571)

[4.38.3: Sequence Flow 68](#_Toc1899848147)

[4.38.4: Additional Arguments 68](#_Toc894005386)

[4.39. Continuous write (Address decrement), continuous read (Address increment) for random three byte locations: (c\_wrDecr\_rdIncr\_byte\_RandL3\_test) 69](#_Toc1311351604)

[4.39.1: Test Purpose 69](#_Toc2045017830)

[4.39.2: Sequences used 69](#_Toc203167638)

[4.39.3: Sequence Flow 69](#_Toc683587512)

[4.39.4: Additional Arguments 70](#_Toc1955481014)

[4.40: Bulk write, bulk read for three random byte locations: (bulk wr\_rd\_byte\_RandL3\_test) 70](#_Toc597713095)

[4.40.1: Test Purpose 70](#_Toc939138355)

[4.40.2: Sequences used 71](#_Toc1479965501)

[4.40.3: Sequence Flow 71](#_Toc1168912155)

[4.40.4: Additional Arguments 71](#_Toc1994748707)

[4.41: Continuous write (Address decrement), continuous read (Address decrement) for three random byte locations: (c\_wr\_rd\_decr\_byte\_RandL3\_test) 72](#_Toc442498338)

[4.41.1: Test Purpose 72](#_Toc992369251)

[4.41.2: Sequences used 72](#_Toc1708014577)

[4.41.3: Sequence Flow 73](#_Toc1799659179)

[4.41.4: Additional Arguments 73](#_Toc563439988)

[4.42: Continuous Write and Read Byte All Location Decrement (Strobe decrement): (c\_wr\_rd\_byte\_ALDecr\_test) 73](#_Toc2075062139)

[4.42.1: Test Purpose 74](#_Toc409983587)

[4.42.2: Sequences Used 74](#_Toc566014581)

[4.42.3: Sequence Flow 74](#_Toc355149776)

[4.42.4: Additional Arguments 74](#_Toc650368275)

[4.43: Continuous write, continuous read (Adress Increment) All Location (Strobe value decrement): (c\_wr\_rd\_incr\_byte\_ALDecr\_test) 75](#_Toc868131480)

[4.43.1: Test Purpose 75](#_Toc2031073253)

[4.43.2: Sequences Used 75](#_Toc688990748)

[4.43.3: Sequence Flow 76](#_Toc950538954)

[4.43.4: Additional Arguments 76](#_Toc1695064008)

[4.44: Continuous write (Address increment), continuous read (Address decrement),All Location Decrement(Strobe value decrement): (c\_wrIncr\_rdDecr\_byte\_AlDecr\_test) 76](#_Toc1943064635)

[4.44.1: Test Purpose 76](#_Toc367013638)

[4.44.2: Sequences Used 76](#_Toc558627395)

[4.44.3: Sequence Flow 77](#_Toc2028393061)

[4.44.4: Additional Arguments 77](#_Toc431250224)

[4.45.1: Test Purpose 77](#_Toc552721173)

[4.45.2: Sequences Used 78](#_Toc1234890389)

[4.45.3: Sequence Flow 78](#_Toc1971966110)

[4.45.4: Additional Arguments 78](#_Toc410535345)

[4.46: Bulk write, bulk read: (bulk\_wr\_rd\_byte\_AlDecr\_test) 79](#_Toc1891664995)

[4.46.1: Test Purpose 79](#_Toc346218603)

[4.46.2: Sequences Used 79](#_Toc816084788)

[4.46.3: Sequence Flow 79](#_Toc1564651173)

[4.46.4: Additional Arguments 80](#_Toc1862312176)

[4.47: Continuous write (Address decrement), continuous read (Address decrement): (c\_wr\_rd\_decr\_byte\_Aldecr\_test) 80](#_Toc83832318)

[4.47.1: Test Purpose 80](#_Toc2001437812)

[4.47.2: Sequences Used 81](#_Toc2113413828)

[4.47.3: Sequence Flow 81](#_Toc1092025892)

[4.47.4: Additional Arguments 81](#_Toc201501007)

[4.48: cont\_wr\_rd\_byte\_ALRandSr\_test 82](#_Toc1647360397)

[4.48.1: Test Purpose: 82](#_Toc1521823929)

[4.48.2: Sequences Used 82](#_Toc921489954)

[4.48.3: Sequence Flow 82](#_Toc2135186187)

[4.48.4: Additional Arguments 83](#_Toc653462153)

[4.49: cont\_wr\_rd\_incr\_byte\_ALRandSr\_test 83](#_Toc687221028)

[4.49.1: Test Purpose 83](#_Toc599505762)

[4.49.2: Sequences Used 83](#_Toc62989120)

[4.49.3: Sequence Flow 84](#_Toc583869861)

[4.49.4: Additional Arguments 84](#_Toc368311487)

[4.50: cont\_wrIncr\_rdDecr\_byte\_ALRandSr\_test 85](#_Toc208440808)

[4.50.1: Test Purpose 85](#_Toc2080355500)

[4.50.2: Sequences Used 85](#_Toc1049511952)

[4.50.3: Sequence Flow 85](#_Toc842820834)

[4.50.4: Additional Arguments 86](#_Toc1068415444)

[4.51: cont\_wrDecr\_rdIncr\_byte\_ALRandSr\_test 86](#_Toc1714558832)

[4.51.1: Test Purpose 87](#_Toc353676738)

[4.51.2: Sequences Used 87](#_Toc2126689636)

[4.51.3: Sequence Flow 87](#_Toc1044828476)

[4.51.4: Additional Arguments 87](#_Toc1602360275)

[4.52: bulk\_wr\_rd\_byte\_ALRandSr\_test 88](#_Toc1728908725)

[4.52.1: Test Purpose 88](#_Toc1854064730)

[4.52.2: Sequences Used 88](#_Toc1300401317)

[4.52.3: Sequence Flow 89](#_Toc223438037)

[4.52.4: Additional Arguments 89](#_Toc80279237)

[4.53: cont\_wr\_rd\_decr\_byte\_ALRandSr\_test 90](#_Toc1941368180)

[4.53.1: Test Purpose 90](#_Toc713263677)

[4.53.2: Sequences Used 90](#_Toc752642558)

[4.53.3: Sequence Flow 90](#_Toc1245386800)

[4.53.4: Additional Arguments 91](#_Toc669864606)

[4.54: cont\_wr\_rd\_byte\_RandL1\_test 92](#_Toc414584766)

[4.54.1: Test Purpose 92](#_Toc1736384805)

[4.54.2: Sequences Used 92](#_Toc1515048679)

[4.54.3: Sequence Flow 92](#_Toc125361593)

[4.54.4: Additional Arguments 93](#_Toc1324801731)

[4.55: cont\_wr\_rd\_incr\_byte\_RandL1\_test 93](#_Toc1480212771)

[4.55.1: Test Purpose 93](#_Toc618252679)

[4.55.2: Sequences Used 93](#_Toc209402896)

[4.55.3: Sequence Flow 94](#_Toc798425581)

[4.55.4: Additional Arguments 94](#_Toc1975909642)

[4.56: cont\_wrIncr\_rdDecr\_byte\_RandL1\_test 95](#_Toc1993954350)

[4.56.1: Test Purpose 95](#_Toc1867199756)

[4.56.2: Sequences Used 95](#_Toc707741218)

[4.56.3: Sequence Flow 95](#_Toc1390387663)

[4.56.4: Additional Arguments 96](#_Toc645759065)

[4. 57: cont\_wrDecr\_rdIncr\_byte\_RandL1\_test 96](#_Toc1051987549)

[4.57.1: Test Purpose 97](#_Toc1484029951)

[4.57.2: Sequences Used 97](#_Toc410369873)

[4.57.3: Sequence Flow 97](#_Toc2118941533)

[4.57.4: Additional Arguments 97](#_Toc1280353836)

[4.58: bulk\_wr\_rd\_byte\_RandL1\_test 98](#_Toc16687553)

[4.58.1: Test Purpose 98](#_Toc1476062766)

[4.58.2: Sequences Used 98](#_Toc1193517652)

[4.58.3: Sequence Flow 99](#_Toc1446043057)

[4.58.4: Additional Arguments 99](#_Toc1569870507)

[4.59: cont\_wr\_rd\_decr\_byte\_RandL1\_test 100](#_Toc1499084405)

[4.59.1: Test Purpose 100](#_Toc945309352)

[4.59.2: Sequences Used 100](#_Toc741497016)

[4.59.3: Sequence Flow 100](#_Toc1970581963)

[4.59.4: Additional Arguments 101](#_Toc1155927514)

[5.Environment 101](#_Toc1567863774)

[5.1: Environment Config 103](#_Toc174566616)

[5.2: Agent 104](#_Toc490821771)

[5.2.1: APB Agent Config 105](#_Toc1248174933)

[5.2.2: APB Defines 106](#_Toc1503558692)

[5.2.3: APB Interface 106](#_Toc1347843052)

[5.2.4: Sequence item 107](#_Toc1203570963)

[5.2.5: Sequencer 109](#_Toc1662207314)

[5.2.6: Sequence 109](#_Toc1825039200)

[5.2.7:Driver 111](#_Toc1864332236)

[5.2.8: Monitor 114](#_Toc123393040)

[5.2.8.1: resetable\_monitor 116](#_Toc422843830)

[5.2.8: Coverage 117](#_Toc792926613)

[5.3 Scoreboard: 119](#_Toc281568630)

# 1.Introduction

In VLSI (Very Large-Scale Integration), a protocol typically refers to a set of rules and conventions governing the communication and interaction between different components or modules within a chip or between chips in a system. These protocols define how data is transmitted, how devices communicate with each other, how transactions are initiated and completed, and how errors are handled. So, protocol verification is crucial for ensur ing the correctness, interoperability, security, performance, and reliability of communication protocols, thereby enabling the development of robust, efficient, and secure systems.

The Advanced Microcontroller Bus Architecture is a standard for designing and developing embedded processors. AMBA helps in modular system design and is highly reusable. The peripherals including timers, UART, PIO, Keypad are of low bandwidth and do not require a pipelined bus interface and AMBA APB, which is also non-pipelined, caters to this need. On the other hand, the CPU(ARM) cores, DMA, high bandwidth memory require a high performance, high bandwidth bus and AMBA AHB caters to this need. All the transitions and transactions are associated with the positive clock edge. Figure 1 shows the AMBA architecture. Notice the presence of two buses, namely AHB and APB buses. As the CPU(ARM) cores, DMA, high bandwidth memory demand high performance, they are connected to AHB bus while the low bandwidth peripherals are interconnected through APB bus. There is an AHB to APB bridge which connects AHB and APB. All APB connected peripherals act as slaves while the AHB-APB bridge (simply APB bridge) acts as the Master and initiates all the transactions. This documentation presents the AMBA APB slave bus protocol design verification strategy using UVM-based custom test bench for AMBA APB verification IP (VIP) development and discusses the results obtained.

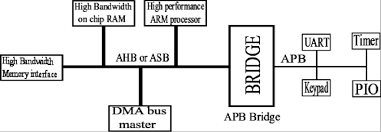


Figure 1: AMBA Bus Architecture

# 2.Methodology

In order to perform design verification, it is essential to know the working and operation of the various components and hierarchy of the AMBA APB Architecture. In this section as displayed in Figure 1, we briefly discuss about the various APB signals, functioning of AMBA APB Master and Slave, the 3 APB Operating states as well as the Read and Write transactions. After having a good understanding of the various details and critical design aspects of the AMBA APB protocol we proceed to implement a suitable verification strategy after understanding the various aspects of UVM and finally, running the regression to obtain and discuss the results.

Result and Conclusion

Simulation

Verification Strategy and Test Plan

UVM

Read and write Transaction

Operating States

APB Master and Slave Signals

AMBA APB Architecture

Figure 2: Methodology of Documentation

## 2.1: APB Master

AMBA that connects to low-level peripheral devices has APB as a part of it. AMBA-APB is made up of two parts: an APB Master/Bridge and a Slave APB, and it is used for connecting a large number of slaves. The APB Bridge transfers data/addresses from the System bus to the APB slave and performs the following tasks.

* It latches the address and remembers it until the transfer is done.
* It figures out where the data should go based on the address and makes particular PSELx high.
* During a switch, only one select signal (PSELx) can be active.
* The data is written to the APB slave in access state.
* When any peripheral wants to read the data from the APB slave, the APB Bridge helps by bringing it back to the system bus.

**APB MASTER**

PWRITE

PSTRB

PADDR

PENABLE

PSEL3

PSEL2

PSEL1

PWDATA

System Bus Slave Interface

PRDATA

PRESETn

PCLK

PSLVERR

Figure 3:APB Master block diagram

## 2.2 APB Slave

The APB Slave is characterized by its simplicity and adaptable interface, making it suitable for integration with multiple slave devices. Its functionality is detailed as follows:

1. It responds to two triggers: the rising edge of the PCLK signal and the detection of a high signal in the PSEL line on the rising edge of PCLK.

2. Similar to the previous point, it also reacts to the condition i.e the rising edge of the PENABLE signal in the presence of a high signal in the PSEL line on the rising edge of PCLK.

3. By incorporating the address (PADDR), select signal (PSELx), write signal (PWRITE) and the strobe signal (PSTRB) the APB Slave determines whether a write operation should update the associated register.

4. During read transfers, the data bus transmits data when the PWRITE signal is low, and both the PENABLE and PSELx signals are high. The address signal (PADDR) facilitates the identification of the register from which data should be retrieved.

5.PSLVERR can be generated depends upon the violation of any condition mentioned in protocol

specification.

PSELx

PENABLE

PSTRB

PADDR

PWRITE

PWDATA

PRESETn

r

PCLK

**APB SLAVE**

PRDATA

PSLVERR

Figure 3:APB Master block diagram

## 2.3. APB Operating States

There are 3 states in the operating status of an APB as listed below.

[1] STATE 1-IDLE

[2] STATE 2-SETUP

[3] STATE 3-ENABLE OR ACCESS

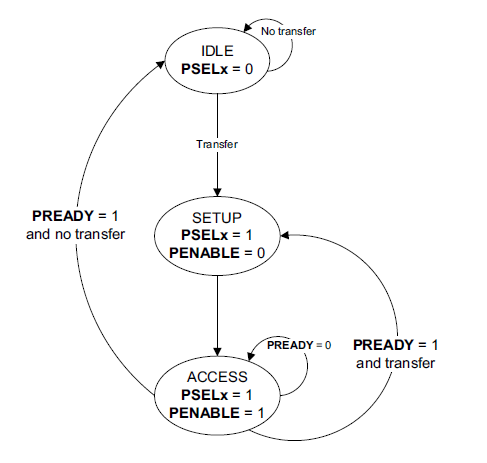


Figure 5:Operating states of APB

The state machine operates through the following states:

**IDLE** This is the default state of the APB interface.

**SETUP** When a transfer is required, the interface moves into the SETUP state, where the appropriate select signal, **PSELx**, is asserted. The interface only remains in the SETUP state for one clock cycle and always moves to the ACCESS state on the next rising edge of the clock.

**ACCESS** The enable signal, **PENABLE**, is asserted in the ACCESS state. The following signals must not change in the transition between SETUP and ACCESS and between cycles in the ACCESS state:

• **PADDR**

• **PWRITE**

• **PWDATA**, only for write transactions

• **PSTRB**

Exit from the ACCESS state is controlled by the **PREADY** signal from the slave.

• If **PREADY** is held LOW by the slave, then the interface remains in the ACCESS state.

• If **PREADY** is driven HIGH by the slave then the ACCESS state is exited and the bus

returns to the IDLE state if no more transfers are required. Alternatively, the bus moves

directly to the SETUP state if another transfer follows.

## 2.4:Write Transfer

This section describes the following types of write transfer:

• With no wait states

• With wait states

### 2.4.1: write with no wait Transfer**:**

The Setup phase of the write transfer occurs at T1 in Figure 6. The select signal, **PSEL**, is asserted, which means that **PADDR**, **PWRITE** and **PWDATA** must be valid. The Access phase of the write transfer is shown at T2 in Figure 6 where **PENABLE** is asserted. **PREADY** is asserted by the slave at the rising edge of **PCLK** to indicate that the write data will be accepted at T3. **PADDR**, **PWDATA**, and any other control signals, must be stable until the transfer completes.

At the end of the transfer, **PENABLE** is deasserted. **PSEL** is also deasserted, unless there is another transfer to the same peripheral.

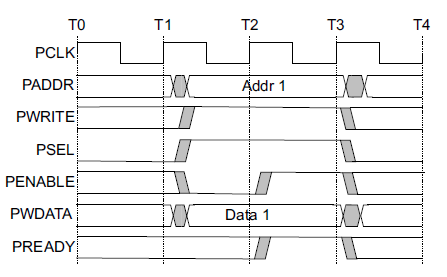


Figure 6:Write with no wait state

### 2.4.2: Write with no wait Transfer:

During an Access phase, when **PENABLE** is HIGH, the Completer extends the transfer by driving **PREADY** LOW. The following signals remain unchanged while **PREADY** remains LOW:

• Address signal, **PADDR**

• Direction signal, **PWRITE**

• Select signal, **PSELx**

• Enable signal, **PENABLE**

• Write data signal, **PWDATA**

• Write strobe signal, **PSTRB**

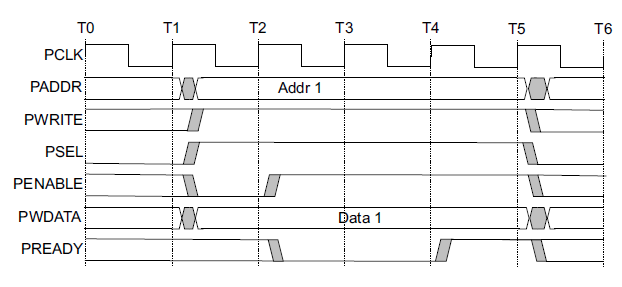


Figure 7: Write with wait transfer

## 2.5: Read Transfer

Two types of read transfer are described in this section:

• With no wait states

• With wait states

### 2.5.1: With no wait states

The timing of the address, **PADDR**, write, **PWRITE**, select, **PSEL**, and enable, **PENABLE**, signals are the same as described in *Write transfers*. The slave must provide the data before the end of the read transfer.

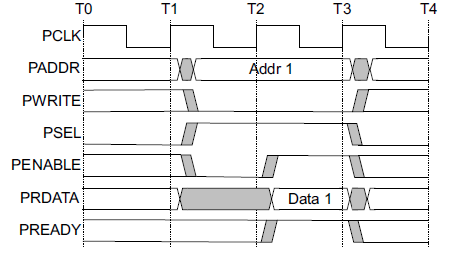


Figure 8: Read transfer with no wait state

### 2.5.2: With wait states

The transfer is extended if **PREADY** is driven LOW during an Access phase. The following signals remain unchanged while **PREADY** remains LOW:

• Address signal, **PADDR**

• Direction signal, **PWRITE**

• Select signal, **PSEL**

• Enable signal, **PENABLE**

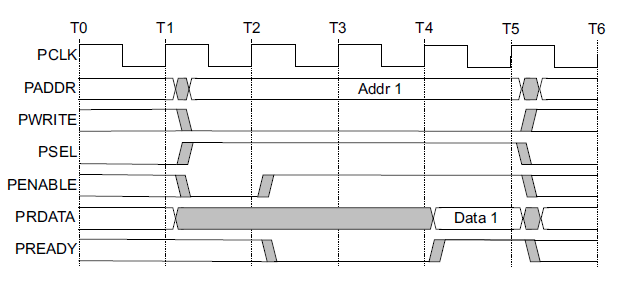


Figure 9: Read transfer with wait state

# 3. UVM (Universal Verification Methodology)

UVM is most commonly used verification methodology for RTL (register transfer level) design. UVM is an extended methodology from System Verilog. It consists of base class libraries. The verification engineer can create different test scenarios by extending these library classes. UVM also provides many other verification features such as factory for object creation, using macros for complex function, reporting mechanism, phasing, configuration etc. We have developed our APB testbench in UVM.

DUT

Interface

AGENT

DRIVER

SEQUENCER

MONITOR

Agent config.

SCOREBOARD & COVERAGE

ENVIRONMENT

Environment Config.

TEST

SEQUENCE LIBRABY

TB\_TOP

## 3.1: TestBench Architecture Overview

Figure 10: Testbench architecture overview

According to our apb testbench the component we used are briefly described below:

# 4: Test

The test is at the top of the hierarchical component that initiates the environment component construction. It is also responsible for the testbench configuration and stimulus generation process. Based on the features and functionalities of the design mentioned in the verification plan, tests are written. The user-defined test class is derived from uvm\_test. The test includes environment class, configurations, and start sequences over sequencers. It is recommended to have a base test that includes all testbench settings including an instance of the environment class, create configurations, etc and varieties of tests as per the verification test plan can be extended from the base test to avoid repeating the same code in every test. The derived tests can also set configurations based on test requirements.

## 4.1: Base Test

First the path of sequencer has been parameterized as APB\_SQCR. The base\_test class is extended from uvm\_test class. As it is a component it has been registered to the factory accordingly. The handle of environment and environment\_config has been taken to control the configuration of lower hierarchical components. To print all the components that has been registered with the factory , a handle of in built uvm\_factory and uvm\_coreservice\_t has been taken. In the constructor function we passed the two arguments of component. Then in the build phase env\_config has been created and some default values of agent configuration have been passed such as by default the agent is active ,it has no coverage and scoreboard . Then the environment\_config was set so that this configuration can be accessed from wherever needed. The env has also been created here. In the end\_of\_elaboration phase the in built uvm\_factory variable named factory is assigned and then is printed . And finally in the run phase a reporting macro is added to see whether this sequence is running or not.

## 4.2: Sanity Simple Test

The sanity\_simple\_test class has been extended from the base\_test. A sanity\_simple\_seq handle has been taken. In the constructor function two arguments have been passed as it is a component. In the run phase the drain time was set as 40 ns so that after dropping objection it still remains in the run phase for 40ns. Then the objection is raised. Sequences are sent for 200 times so for loop is used accordingly. Then the sequence has been created and started. Finally the objection is dropped.

### 4.2.1: Test Purpose

The purpose of the test is to send a simple reset and directed write and read sequence to verify if the dut is responding correctly according to the protocol.

### 4.2.2: Sequences Used

The main sequence that has been used in this test is sanity\_simple\_seq. In the sanity\_simple\_seq which is derived form apb\_base\_seq, the handles of apb\_reset\_seq as apb\_rst and apb\_directed\_seq as apb\_seq[2] have been taken. In the body task of this sequence first apb\_rst sequence has been created and started. After 20 ns directed sequence apb\_seq[0] has been created and some directed value has been assigned to write some value in a random address. Then this sequence has been started. Right after 20ns we created apb\_seq[1] and read the value in the same address that was written previously. Then this sequence is started. If the write and read value are same then this test will be considered failed otherwise passed.

### 4.2.3: Sequence Flow

1.sanity\_simple\_seq

1.1.apb\_reset\_seq

1.2:apb\_directed\_seq[0]

1.3:apb\_directed\_seq[1]

### 4.2.4: Additioal Arguments

No other additional arguments have been used to run the test.

## 4.3: Error Test

Error\_test has been extended from base\_test

**Fields:**

apb\_reset\_seq apb\_rst

error\_drive\_seq error\_drive

**Methods:**

**1.Constructor :** function new**.** As this is a component there are two arguments named string name and uvm\_component.

**2.run\_phase:** virtual task. Drain time is set to 50 ns. Then objection is raised. First apb\_rst sequence is created and then started in the sequencer parameterized as `APB\_SQCR. Then the error\_drive sequence is created and started for 20 times. Finally the phase objection is dropped.

### 4.3.1: Test Purpose

Detect error by providing intentional error driving data.

### 4.3.2: Sequences used

For this test a specific sequence has been created which is error\_drive\_seq. A handle of apb\_seq\_item is taken. In the body task of this sequence item is created. PRESETn\_i,PENABLE\_i, PSEL\_i are set to 1. PADDR\_i, PWDATA\_i are set to random whereas PWRITE\_i and PSTRB\_i are assigned random value between 0 and 1. Then finish\_item is called. Another sequence used in this test is apb\_rst.

### 4.3.3: Sequence Flow

1.apb\_reset\_seq

2.error\_drive\_seq

### 4.3.4: Additional Arguments

A specific argument needs to be passed to run this test from command line. There are five error driver in this testbench. One of them should be passed when running the test. Considering a makefile, the command would be like this(make xsim TEST\_NAME=error\_test ERR= sel\_enb\_error\_drive/sel\_enb\_high\_drive/enb\_high\_bef\_sel\_drive/enb\_high\_change\_ctrl\_drive/enb\_low\_before\_pready\_drive)

## 4.4: Continuous Write and Read Word Test(c\_wr\_rd\_word\_test)

### 4.4.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing and reading operations using strobe `1111` for the entire 32-bit address bus. This test ensures that data can be continuously written and then continuously read back correctly, confirming the basic read and write capabilities of the system. The test will pass if the data written to each address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.4.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.4.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.4.4: Additional Arguments

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test”,

+UVM\_TIMEOUT=500000”,

+UVM\_VERBOSITY=UVM\_HIGH”,

+WORD\_KNOB=100”,

+HALF\_WORD\_KNOB=0”,

+BYTE\_KNOB=0",

+WR\_ADDR= ",

+RD\_ADDR= ",

+LOC=rand",

**owner:** Rashid + Limon"

## 4.5: Continuous write (Address increment), continuous read (Address increment): (c\_wr\_rd\_incr\_word\_test)

### 4.5.1: Test Purpose

The purpose of the test is to ensure proper functionality when continuously writing with address increment and then continuously reading with address increment using strobe `1111`. This test checks that the system can handle sequential addressing, confirming that data is correctly written to and read from successive memory locations. The test will pass if the data written to each incremented address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.5.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.5.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.5.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=100",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=0",

+WR\_ADDR=incr ",

+RD\_ADDR=incr ",

+LOC=rand",

**owner:** Rashid + Limon"

## 4.6: Continuous write (Address increment), continuous read (Address decrement): (c\_wrIncr\_rdDecr\_word\_test)

### 4.6.1: Test Purpose

The purpose of the test is to test the system's ability to handle continuous writing with address increments and continuous reading with address decrements using strobe `1111`. This test ensures that the system can correctly write data to successive memory locations and read back the data in reverse order, verifying the robustness of address handling. The test will pass if the data written to each incremented address is correctly read back in reverse order and matches the expected values. Failures can be checked in the scoreboard.

### 4.6.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.6.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.6.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=100",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=0",

+WR\_ADDR=incr",

+RD\_ADDR=decr",

+LOC=rand",

**owner:** Rashid + Limon"

## 4.7: Continuous write (Address decrement), continuous read (Address increment): (c\_wrDecr\_rdIncr\_word\_test)

### 4.7.1: Test Purpose

The purpose of the test is to verify the system's behavior during continuous writing with address decrements and continuous reading with address increments using strobe `1111`. This test checks that the system can correctly write data to memory locations in reverse order and read back the data sequentially, verifying proper functionality of address handling. The test will pass if the data written to each decremented address is correctly read back in sequential order and matches the expected values. Failures can be checked in the scoreboard.

### 4.7.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.7.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.7.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=100",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=0",

+WR\_ADDR=decr",

+RD\_ADDR=incr",

+LOC=rand",

**owner:** Rashid + Limon"

## 4.8: Bulk write, bulk read(bulk\_wr\_rd\_word\_test)

### 4.8.1: Test Purpose

The purpose of the test is to test the performance and correctness of bulk write and read operations using strobe 1111. This test ensures that the system can handle large amounts of data being written and read in a single operation, verifying the efficiency and reliability of bulk data transfers. During the bulk test, all variables are generated randomly, and the memory size can be increased using KNOBS at runtime. The test will pass if all bulk data written is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.8.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.8.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.8.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

"run\_op : +UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=100",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=0",

+WR\_ADDR= ",

+RD\_ADDR= ",

+LOC=RandL1",

**owner:** Rashid + Limon"

## 4.9: Continuous write (Address decrement), continuous read (Address decrement): (c\_wr\_rd\_decr\_word\_test)

### 4.9.1: Test Purpose

The purpose of the test is to ensure proper functionality during continuous writing and reading operations with address decrements using strobe `1111`. This test verifies that the system can correctly handle data writes and reads when the address decreases, confirming the system's ability to manage reverse addressing. The test will pass if the data written to each decremented address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.9.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.9.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.9.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=100",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=0",

+WR\_ADDR=decr ",

+RD\_ADDR=decr " ,

+LOC=rand",

**owner:** Rashid + Limon"

**#### Half-Word Both Location (BL) Tests**

## 4.10: Continuous write and read for both locations (1100, 0011): (c\_wr\_rd\_Hword\_BL\_test)

### 4.10.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing and reading operations using strobe `1100` and `0011` both in same address. This test ensures that data can be continuously written and read correctly for both half-word locations, confirming the proper handling of half-word data. The test will pass if the data written to each half-word location (1100 followed by 0011) is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.10.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.10.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.10.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=100",

+BYTE\_KNOB=0",

+WR\_ADDR= ",

+RD\_ADDR= ",

+LOC=ALincr",

**owner:** Rashid + Limon"

## 4.11: Continuous write (Address increment), continuous read (Address increment) for both locations (1100, 0011): (c\_wr\_rd\_incr\_Hword\_BL\_test)

### 4.11.1: Test Purpose

The purpose of the test is to ensure proper functionality when continuously writing and reading with address increment using strobe `1100` and `0011` both in same address. This test checks that the system can handle sequential addressing for half-word data at both locations. The test will pass if the data written to each incremented address for half-word locations (1100 followed by 0011) is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.11.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.11.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.11.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=100",

+BYTE\_KNOB=0",

+WR\_ADDR=incr ",

+RD\_ADDR=incr " ,

+LOC=ALincr",

**owner:** Rashid + Limon"

## 4.12. Continuous write (Address increment), continuous read (Address decrement) for both locations (1100, 0011): c\_wrIncr\_rdDecr\_Hword\_BL\_test

### 4.12.1: Test Purpose

The purpose of the test is to test the system's ability to handle continuous writing with address increments and continuous reading with address decrements using strobe `1100` and `0011` both in same address. This test ensures that the system can correctly manage half-word data writes to successive addresses and reads in reverse order, verifying address handling robustness. The test will pass if the data written to each incremented address for half-word locations (1100 followed by 0011) is correctly read back in reverse order and matches the expected values. Failures can be checked in the scoreboard.

### 4.12.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.12.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.12.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=100",

+BYTE\_KNOB=0",

+WR\_ADDR=incr ",

+RD\_ADDR=decr ",

+LOC=ALincr",

**owner:** Rashid + Limon"

## 4.13: Continuous write (Address decrement), continuous read (Address increment) for both locations (1100, 0011):( c\_wrDecr\_rdIncr\_Hword\_BL\_test)

### 4.13.1: Test Purpose

The purpose of the test is to verify the system's behavior during continuous writing with address decrements and continuous reading with address increments using strobe `1100` and `0011` both in same address. This test checks that the system can correctly manage half-word data writes to memory locations in reverse order and reads sequentially, verifying proper functionality of address handling. The test will pass if the data written to each decremented address for half-word locations (1100 followed by 0011) is correctly read back in sequential order and matches the expected values. Failures can be checked in the scoreboard.

### 4.13.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.13.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.13.4: Additional Arguments

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=100",

+BYTE\_KNOB=0",

+WR\_ADDR=decr",

+RD\_ADDR=incr ",

+LOC=ALincr",

**owner:** Rashid + Limon"

## 4.14: Bulk write, bulk read for both locations (1100, 0011): (bulk\_wr\_rd\_Hword\_BL\_test)

### 4.14.1: Test Purpose

The purpose of the test is to test the performance and correctness of bulk write and read operations using strobe 1111. This test ensures that the system can handle large amounts of data being written and read in a single operation, verifying the efficiency and reliability of bulk data transfers. During the bulk test, all variables are generated randomly, and the memory size can be increased using KNOBS at runtime. The test will pass if all bulk data written is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.14.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.14.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.14.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=100",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=0",

+WR\_ADDR= ",

+RD\_ADDR= " ,

+LOC=ALincr",

**owner:** Rashid + Limon"

## 4.15: Continuous write (Address decrement), continuous read (Address decrement) for both locations (1100, 0011): (c\_wr\_rd\_decr\_Hword\_BL\_test)

### 4.15.1: Test Purpose

The purpose of the test is to ensure proper functionality during continuous writing and reading operations with address decrements using strobe `1100` and `0011 both in same address. This test verifies that the system can correctly handle half-word data writes and reads when the address decreases, confirming the system's ability to manage reverse addressing for half-words. The test will pass if the data written to each decremented address for both half-word locations (1100 followed by 0011) is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.15.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.15.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.15.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=100",

+BYTE\_KNOB=0",

+WR\_ADDR=decr ",

+RD\_ADDR=decr " ,

+LOC=ALincr",

**owner:** Rashid + Limon"

**Half-Word One Location (L1) Tests:**

## 4.16: Continuous write and read for location 1 (1100 or 0011): (c\_wr\_rd\_Hword\_L1\_test)

### 4.16.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing and reading operations using strobe 1100 or 0011 for half-word location 1. This test ensures that data can be continuously written to and read from the address bus, confirming proper handling of half-word data at one location. The test will pass if the data written to each half-word location 1 (1100 or 0011) is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.16.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.16.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.16.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=100",

+BYTE\_KNOB=0",

+WR\_ADDR= ",

+RD\_ADDR= " ,

+LOC=RandL1",

**owner:** Rashid + Limon"

## 4.17: Continuous write (Address increment), continuous read (Address increment) for location 1 (1100 or 0011): (c\_wr\_rd\_incr\_Hword\_L1\_test)

### 4.17.1: Test Purpose

The purpose of the test is to ensure proper functionality when continuously writing and reading with address increment using strobe 1100 or 0011 for half-word location 1. This test checks that the system can handle sequential addressing for half-word data at one location. The test will pass if the data written to each incremented address for half-word location 1 (1100 or 0011) is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.17.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.17.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.17.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=100",

+BYTE\_KNOB=0",

+WR\_ADDR=incr ",

+RD\_ADDR=incr " ,

+LOC=RandL1",

**owner :** Rashid + Limon"

## 4.18: Continuous write (Address increment), continuous read (Address decrement) for location 1 (1100 or 0011): (c\_wrIncr\_rdDecr\_Hword\_L1\_test.sv)

### 4.18.1: Test Purpose

The purpose of the test is to test the system's ability to handle continuous writing with address increments and continuous reading with address decrements using strobe 1100 or 0011 for half-word location 1. This test ensures that the system can correctly manage half-word data writes to successive addresses and reads in reverse order, verifying address handling robustness. The test will pass if the data written to each incremented address for half-word location 1 (1100 or 0011) is correctly read back in reverse order and matches the expected values. Failures can be checked in the scoreboard.

### 4.18.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.18.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.18.4: Additional Arguments

run\_op :

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=100",

+BYTE\_KNOB=0",

+WR\_ADDR=incr ",

+RD\_ADDR=decr " ,

+LOC=RandL1",

**owner:** Rashid + Limon"

## 4.19: Continuous write (Address decrement), continuous read (Address increment) for location 1 (1100 or 0011) : (c\_wrDecr\_rdIncr\_Hword\_L1\_test)

### 4.19.1: Test Purpose

The purpose of the test is to verify the system's behavior during continuous writing with address decrements and continuous reading with address increments using strobe 1100 or 0011 for half-word location 1. This test checks that the system can handle half-word data writes to addresses in reverse order and reads sequentially, ensuring proper decrementing and incrementing of addresses. The test will pass if the data written to each decremented address for half-word location 1 (1100 or 0011) is correctly read back in sequential order and matches the expected values. Failures can be checked in the scoreboard.

### 4.19.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.19.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.19.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=100",

+BYTE\_KNOB=0",

+WR\_ADDR=decr",

+RD\_ADDR=incr ",

+LOC=RandL1",

**owner:** Rashid + Limon"

## 4.20: Bulk write, bulk read for location 1 (1100 or 0011): (bulk\_wr\_rd\_Hword\_L1\_test)

### 4.20.1: Test Purpose

The purpose of the test is to test the performance and correctness of bulk write and read operations using strobe 1100 or 0011 for half-word location 1. This test ensures that the system can manage large volumes of half-word data written to and read from either location efficiently, confirming the reliability of bulk data transfers. The test will pass if all bulk data written to half-word location 1 (1100 or 0011) is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.20.2: Sequences Used

No specific sequences detailed.

### 4.20.3: Sequence Flow

No specific sequence flow detailed.

### 4.20.4: Additional Arguments

No other additional arguments have been used to run the test.

## 4.21: Continuous write (Address decrement), continuous read (Address decrement) for location 1 (1100 or 0011): (c\_wr\_rd\_decr\_Hword\_L1\_test)

### 4.21.1: Test Purpose

The purpose of the test is to ensure proper functionality during continuous writing and reading operations with address decrements using strobe 1100 or 0011 for half-word location 1. This test verifies that the system can correctly handle half-word data writes and reads when the address decreases, confirming the system's ability to manage reverse addressing for one location. The test will pass if the data written to each decremented address for half-word location 1 (1100 or 0011) is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.21.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.21.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.21.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=100",

+BYTE\_KNOB=0",

+WR\_ADDR=decr",

+RD\_ADDR=incr ",

+LOC=RandL1",

**owner :** Rashid + Limon"

## 4.22: Bulk write, bulk read for both locations (1100, 0011) in random order : (bulk\_wr\_rd\_Hword\_BLRandSr\_test)

### 4.22.1: Test Purpose

The purpose of the test is to test the performance and correctness of bulk write and read operations using strobe 1111. This test ensures that the system can handle large amounts of data being written and read in a single operation, verifying the efficiency and reliability of bulk data transfers. During the bulk test, all variables are generated randomly, and the memory size can be increased using KNOBS at runtime. The test will pass if all bulk data written is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.22.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.22.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.22.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=100",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=0",

+WR\_ADDR= ",

+RD\_ADDR= " ,

+LOC=RandL1",

**owner :** Rashid + Limon"

## 4.23: Continuous write (Address decrement), continuous read (Address decrement) for location 1 (1100 or 0011): (c\_wr\_rd\_decr\_Hword\_L1\_test)

### 4.23.1: Test Purpose

The purpose of the test is to ensure proper functionality during continuous writing and reading operations with address decrements using strobe 1100 or 0011 for half-word location 1. This test verifies that the system can correctly handle half-word data writes and reads when the address decreases, confirming the system's ability to manage reverse addressing for one location. The test will pass if the data written to each decremented address for half-word location 1 (1100 or 0011) is correctly read back and matches the expected values. Failures can be checked in the scoreboard

### 4.23.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.23.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.23.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=100",

+BYTE\_KNOB=0",

+WR\_ADDR=decr ",

+RD\_ADDR=decr " ,

+LOC=RandL1",

**owner :** Rashid + Limon"

**#### ByteTests**

## 4.24: Continuous Write Read Byte All Location Increase Test(0001,0010,0100,1000): (c\_wr\_rd\_byte\_ALincr\_test)

### 4.24.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing and reading operations using strobe `(0001,0010,0100,1000) in same address in an increasing manner. This test ensures that data can be continuously written and read correctly for all the byte locations, confirming the proper handling of byte data. The test will pass if the data written to each byte location 1000,0100,0010,0001 in a specific address is correctly read back and matches the expected values and failures will be caused if the read and written data don’t match.

### 4.24.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_byte\_ALinc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.24.3: Sequence Flow

1. `cont\_wr\_rd\_byte\_ALinc\_seq`

1.1. `apb\_reset\_seq` (rst\_seq)

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.24.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_byte\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR= ",

+RD\_ADDR= ",

+LOC=ALincr",

**owner:** orthy

## 4.25: Continuous write (Address increment), continuous read (Address increment) for all byte locations (0001,0010,0100,100): (c\_wr\_rd\_incr\_byte\_ALIncr\_test)

### 4.25.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing and reading operations using strobe `(0001,0010,0100,1000) in same address in an increasing manner. This test ensures that data can be continuously written and read correctly for all the byte locations, confirming the proper handling of byte data. The test will pass if the data written to each byte location 1000,0100,0010,0001 in a specific address is correctly read back and matches the expected values and failures will be caused if the read and written data don’t match.

### 4.25.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_byte\_ALinc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.25.3: Sequence Flow

1. `cont\_wr\_rd\_byte\_ALinc\_seq`

1.1. `apb\_reset\_seq` (rst\_seq)

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.25.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_byte\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR= incr",

+RD\_ADDR= incr",

+LOC=ALincr",

**owner:** orthy

## 4.26. Continuous write (Address increment), continuous read (Address decrement) for both all byte locations (0001,0010,0100,1000): c\_wrIncr\_rdDecr\_byte\_ALIncr\_test

### 4.26.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing and reading operations using strobe `(0001,0010,0100,1000) in same address in an increasing manner. This test ensures that data can be continuously written and read correctly for all the byte locations, confirming the proper handling of byte data. The test will pass if the data written to each byte location 1000,0100,0010,0001 in a specific address is correctly read back and matches the expected values and failures will be caused if the read and written data don’t match.

### 4.26.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_byte\_ALinc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.26.3: Sequence Flow

1. `cont\_wr\_rd\_byte\_ALinc\_seq`

1.1. `apb\_reset\_seq` (rst\_seq)

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.26.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_byte\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR= incr",

+RD\_ADDR= decr",

+LOC=ALincr",

**owner:** orthy

### 

## 4.27: Continuous write (Address decrement), continuous read (Address increment) for all byte locations (0001,0010,0100,1000): (c\_wrDecr\_rdIncr\_byte\_ALIncr\_test)

### 4.27.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing and reading operations using strobe `(0001,0010,0100,1000) in same address in an increasing manner. This test ensures that data can be continuously written and read correctly for all the byte locations, confirming the proper handling of byte data. The test will pass if the data written to each byte location 1000,0100,0010,0001 in a specific address is correctly read back and matches the expected values and failures will be caused if the read and written data don’t match.

### 4.27.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_byte\_ALinc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.27.3: Sequence Flow

1. `cont\_wr\_rd\_byte\_ALinc\_seq`

1.1. `apb\_reset\_seq` (rst\_seq)

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.27.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_byte\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR= decr",

+RD\_ADDR= incr",

+LOC=ALincr",

**owner:** orthy

### 

## 4.28: Bulk write, bulk read for all byte locations (0001,0010,0100,1000): (bulk\_wr\_rd\_byte\_ALincr\_test)

### 4.28.1: Test Purpose

The purpose of the test is to test the performance and correctness of bulk write and read operations using strobe 1111. This test ensures that the system can handle large amounts of data being written and read in a single operation, verifying the efficiency and reliability of bulk data transfers. During the bulk test, all variables are generated randomly, and the memory size can be increased using KNOBS at runtime. The test will pass if all bulk data written is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.28.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_byte\_ALinc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.28.3: Sequence Flow

1. `cont\_wr\_rd\_byte\_ALinc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.28.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR= ",

+RD\_ADDR= " ,

+LOC=ALincr",

**owner:** orthy

## 4.29: Continuous write (Address decrement), continuous read (Address decrement) for all byte locations (0001,0010,0100,1000): (c\_wr\_rd\_decr\_byte\_ALIncr\_test)

### 4.29.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing and reading operations using strobe `(0001,0010,0100,1000) in same address in an increasing manner. This test ensures that data can be continuously written and read correctly for all the byte locations, confirming the proper handling of byte data. The test will pass if the data written to each byte location 1000,0100,0010,0001 in a specific address is correctly read back and matches the expected values and failures will be caused if the read and written data don’t match.

### 4.29.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_byte\_ALinc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.29.3: Sequence Flow

1. `cont\_wr\_rd\_byte\_ALinc\_seq`

1.1. `apb\_reset\_seq` (rst\_seq)

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.29.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_byte\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR= decr",

+RD\_ADDR= decr",

+LOC=ALincr",

**owner:** orthy

## 4.30: Continuous write and read for two random byte location RandL2(0010): (c\_wr\_rd\_byte\_RandL2\_test)

### 4.30.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing and reading operations using 2 random strobe location for the entire 32-bit address bus. This test ensures that data can be continuously written and then continuously read back correctly, confirming the basic read and write capabilities of the system. The test will pass if the data written to each address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.30.2: Sequences used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.30.3: Sequence Flow

1.cont\_wr\_rd\_knobs\_loc\_seq`

2.`apb\_reset\_seq`

3.`apb\_directed\_seq` (wr\_seq)

4.`apb\_directed\_seq` (rd\_seq)

### 4.30.4: Additional Arguments

"run\_op :

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

: +WORD\_KNOB=0",

: +HALF\_WORD\_KNOB=0",

: +BYTE\_KNOB=100",

: +WR\_ADDR= ",

: +RD\_ADDR= ",

: +LOC=RandL2"

owner:

## 4.31: Continuous write (Address increment), continuous read (Address increment) for two random byte location: (c\_wr\_rd\_incr\_byte\_RandL2\_test)

### 4.31.1: Test Purpose

The purpose of the test is to ensure proper functionality when continuously writing with address increment and then continuously reading with address increment using strobe 2 random location. This test checks that the system can handle sequential addressing, confirming that data is correctly written to and read from successive memory locations. The test will pass if the data written to each incremented address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.31.2: Sequences used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.31.3: Sequence Flow

1.`cont\_wr\_rd\_knobs\_loc\_seq`

2. `apb\_reset\_seq`

3. `apb\_directed\_seq` (wr\_seq)

4. `apb\_directed\_seq` (rd\_seq)

### 4.31.4: Additional Arguments

"run\_op :

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=incr",

+RD\_ADDR=incr" ,

+LOC=RandL2"

owner:orthy

## 4.32: Continuous write (Address increment), continuous read (Address decrement) for two Random byte location: (c\_wrIncr\_rdDecr\_byte\_RandL2\_test)

### 4.32.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing address increament and reading address decrement operations using 2 random strobe location for the entire 32-bit address bus. This test ensures that data can be continuously written and then continuously read back correctly, confirming the basic read and write capabilities of the system. The test will pass if the data written to each address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.32.2: Sequences used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.32.3: Sequence Flow

1.`cont\_wr\_rd\_knobs\_loc\_seq`

2. `apb\_reset\_seq`

3. `apb\_directed\_seq` (wr\_seq)

4. `apb\_directed\_seq` (rd\_seq)

### 4.32.4: Additional Arguments

"run\_op :

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=incr",

+RD\_ADDR=decr" ,

+LOC=RandL2"

## 4.33: - Continuous write (Address decrement), continuous read (Address increment) for two random byte location: (c\_wrDecr\_rdIncr\_byte\_RandL2\_test)

### 4.33.1: Test Purpose

The purpose of the test is to verify the system's behavior during continuous writing with address decrements and continuous reading with address increments using strobe in random 2 location (1000,0100), (1000,0010), (0100,1000)..... This test checks that the system can correctly write data to memory locations in reverse order and read back the data sequentially, verifying proper functionality of address handling. The test will pass if the data written to each decremented address is correctly read back in sequential order and matches the expected values. Failures can be checked in the scoreboard.

### 4.33.2: Sequences used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.33.3: Sequence Flow

1.`cont\_wr\_rd\_knobs\_loc\_seq`

2. `apb\_reset\_seq`

3. `apb\_directed\_seq` (wr\_seq)

4. `apb\_directed\_seq` (rd\_seq)

### 4.33.4: Additional Arguments

"run\_op :

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=decr",

+RD\_ADDR=incr" ,

+LOC=RandL2"

## 4.34: Bulk write, bulk read for two random byte locations: (bulk wr\_rd\_byte\_RandL2\_test)

### 4.34.1: Test Purpose

The purpose of the test is to test the performance and correctness of bulk write and read operations using strobe in 2 random locations. This test ensures that the system can handle large amounts of data being written and read in a single operation, verifying the efficiency and reliability of bulk data transfers. During the bulk test, all variables are generated randomly, and the memory size can be increased using KNOBS at runtime. The test will pass if all bulk data written is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.34.2: Sequences used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.34.4: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

2. `apb\_reset\_seq`

3. `apb\_directed\_seq` (wr\_seq)

4. `apb\_directed\_seq` (rd\_seq)

### 4.34.4: Additional Arguments

"run\_op :

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+BYTE\_KNOB=100",

+LOC=RandL2",

+WR\_ADDR= ",

+RD\_ADDR= "

## 4.35: Continuous write (Address decrement), continuous read (Address decrement) for two random byte locations: (c\_wr\_rd\_decr\_byte\_RandL2\_test)

### 4.35.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing and reading address decrement operations using 2 random strobe location for the entire 32-bit address bus. This test ensures that data can be continuously written and then continuously read back correctly, confirming the basic read and write capabilities of the system. The test will pass if the data written to each address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.35.2: Sequences used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.35.3: Sequence Flow

1.`cont\_wr\_rd\_knobs\_loc\_seq`

2. `apb\_reset\_seq`

3. `apb\_directed\_seq` (wr\_seq)

4. `apb\_directed\_seq` (rd\_seq)

### 4.35.4: Additional Arguments

"run\_op :

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=decr",

+RD\_ADDR=decr" ,

+LOC=RandL2"

##RandL3##

## 4.36: Continuous write and read for random three byte locations: (c\_wr\_rd\_byte\_RandL3\_test)

### 4.36.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing and reading operations using 3 random strobe location for the entire 32-bit address bus. This test ensures that data can be continuously written and then continuously read back correctly, confirming the basic read and write capabilities of the system. The test will pass if the data written to each address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.36.2: Sequences used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.36.3: Sequence Flow

1.`cont\_wr\_rd\_knobs\_loc\_seq`

2. `apb\_reset\_seq`

3. `apb\_directed\_seq` (wr\_seq)

4. `apb\_directed\_seq` (rd\_seq)

### 4.36.4: Additional Arguments

"run\_op :

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0"

+BYTE\_KNOB=100",

+WR\_ADDR= ",

+RD\_ADDR= ",

+LOC=RandL3"

## 4.37: Continuous write (Address increment), continuous read (Address increment) for random three byte locations(c\_wr\_rd\_incr\_byte\_RandL3\_test)

### 4.37.1: Test Purpose

The purpose of the test is to ensure proper functionality when continuously writing with address increment and then continuously reading with address increment using strobe 3 random location. This test checks that the system can handle sequential addressing, confirming that data is correctly written to and read from successive memory locations. The test will pass if the data written to each incremented address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.37.2: Sequences used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.37.3: Sequence Flow

1.`cont\_wr\_rd\_knobs\_loc\_seq`

2. `apb\_reset\_seq`

3. `apb\_directed\_seq` (wr\_seq)

4. `apb\_directed\_seq` (rd\_seq)

### 4.37.4: Additional Arguments

"run\_op :

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=incr",

+RD\_ADDR=incr" ,

"run\_op : +LOC=RandL3"

## 4.38: Continuous write (Address increment), continuous read (Address decrement) for random three byte locations: (c\_wrIncr\_rdDecr\_byte\_RandL3\_test)

### 4.38.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing address increament and reading address decrement operations using 3 random strobe location for the entire 32-bit address bus. This test ensures that data can be continuously written and then continuously read back correctly, confirming the basic read and write capabilities of the system. The test will pass if the data written to each address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.38.2: Sequences used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.38.3: Sequence Flow

1.`cont\_wr\_rd\_knobs\_loc\_seq`

2. `apb\_reset\_seq`

3. `apb\_directed\_seq` (wr\_seq)

4. `apb\_directed\_seq` (rd\_seq)

### 4.38.4: Additional Arguments

"run\_op :

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=incr",

+RD\_ADDR=decr" ,

+LOC=RandL3"

## 4.39. Continuous write (Address decrement), continuous read (Address increment) for random three byte locations: (c\_wrDecr\_rdIncr\_byte\_RandL3\_test)

### 4.39.1: Test Purpose

The purpose of the test is to verify the system's behavior during continuous writing with address decrements and continuous reading with address increments using strobe in random 3 location (1000,0100,0010),(0001,1000,0010),(0100,1000,0001)..... This test checks that the system can correctly write data to memory locations in reverse order and read back the data sequentially, verifying proper functionality of address handling. The test will pass if the data written to each decremented address is correctly read back in sequential order and matches the expected values. Failures can be checked in the scoreboard.

### 4.39.2: Sequences used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.39.3: Sequence Flow

1.`cont\_wr\_rd\_knobs\_loc\_seq`

2. `apb\_reset\_seq`

3. `apb\_directed\_seq` (wr\_seq)

4. `apb\_directed\_seq` (rd\_seq)

### 4.39.4: Additional Arguments

"run\_op :

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=decr",

+RD\_ADDR=incr" ,

+LOC=RandL3"

## 4.40: Bulk write, bulk read for three random byte locations: (bulk wr\_rd\_byte\_RandL3\_test)

### 4.40.1: Test Purpose

The purpose of the test is to test the performance and correctness of bulk write and read operations using strobe in 3 random locations. This test ensures that the system can handle large amounts of data being written and read in a single operation, verifying the efficiency and reliability of bulk data transfers. During the bulk test, all variables are generated randomly, and the memory size can be increased using KNOBS at runtime. The test will pass if all bulk data written is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.40.2: Sequences used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.40.3: Sequence Flow

1.`cont\_wr\_rd\_knobs\_loc\_seq`

2. `apb\_reset\_seq`

3. `apb\_directed\_seq` (wr\_seq)

4. `apb\_directed\_seq` (rd\_seq)

### 4.40.4: Additional Arguments

"run\_op :

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+BYTE\_KNOB=100",

+LOC=RandL3",

+WR\_ADDR= ",

+RD\_ADDR= "

## 4.41: Continuous write (Address decrement), continuous read (Address decrement) for three random byte locations: (c\_wr\_rd\_decr\_byte\_RandL3\_test)

### 4.41.1: Test Purpose

The purpose of the test is to verify the functionality of continuous writing and reading address decrement operations using 3 random strobe location for the entire 32-bit address bus. This test ensures that data can be continuously written and then continuously read back correctly, confirming the basic read and write capabilities of the system. The test will pass if the data written to each address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.41.2: Sequences used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.41.3: Sequence Flow

1.`cont\_wr\_rd\_knobs\_loc\_seq`

2. `apb\_reset\_seq`

3. `apb\_directed\_seq` (wr\_seq)

4. `apb\_directed\_seq` (rd\_seq)

### 4.41.4: Additional Arguments

"run\_op :

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=decr",

+RD\_ADDR=decr" ,

+LOC=RandL3"

## 4.42: Continuous Write and Read Byte All Location Decrement (Strobe decrement): (c\_wr\_rd\_byte\_ALDecr\_test)

### 4.42.1: Test Purpose

The purpose of this test is to verify the functionality of continuous writing and reading operations using a decrementing strobe pattern of `1000`, `0100`, `0010`, and `0001` from a random point. During the test, data is written to random locations and read from random locations, with the strobe pattern decreasing at each step. The test is considered successful if the data written to each address is correctly read back and matches the expected values. Any discrepancies or failures can be identified and checked in the scoreboard.

### 4.42.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.42.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.42.4: Additional Arguments

**"run\_op :**

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR= ",

+RD\_ADDR= " ,

+LOC= Aldecr",

"owner : bappi"

## 4.43: Continuous write, continuous read (Adress Increment) All Location (Strobe value decrement): (c\_wr\_rd\_incr\_byte\_ALDecr\_test)

### 4.43.1: Test Purpose

The purpose of this test is to verify the functionality of continuous writing and reading operations by incrementing the address from a random starting point while decrementing the strobe pattern (`1000`, `0100`, `0010`, `0001`). In this test, data is written to random locations and read from random locations, with the strobe pattern decreasing at each step. The test is deemed successful if the data written to each address is correctly read back and matches the expected values. Any failures or discrepancies can be identified and analyzed in the scoreboard.

### 4.43.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.43.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.43.4: Additional Arguments

**"run\_op :**

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=incr ",

+RD\_ADDR=incr " ,

+LOC=Aldecr",

"owner : bappi"

## 4.44: Continuous write (Address increment), continuous read (Address decrement),All Location Decrement(Strobe value decrement): (c\_wrIncr\_rdDecr\_byte\_AlDecr\_test)

### 4.44.1: Test Purpose

The purpose of this test is to verify the functionality of continuous writing and reading operations by incrementing the write address from a random starting point while decrementing the read address, all while decrementing the strobe pattern (`1000`, `0100`, `0010`, `0001`). In this test, data is written to random locations and read from random locations, with the strobe pattern decreasing at each step. The test is considered successful if the data written to each address is correctly read back and matches the expected values. Any failures or discrepancies can be identified and analyzed in the scoreboard.

### 4.44.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.44.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.44.4: Additional Arguments

**"run\_op :**

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=incr",

+RD\_ADDR=decr" ,

+LOC=Aldecr"

"owner : bappi"

**4.45: Continuous write (Address decrement), continuous read (Address increment),All Location Decrement(Strobe value Decrement): (c\_wrDecr\_rdIncr\_byte\_AlDecr\_test)**

### 4.45.1: Test Purpose

The purpose of this test is to verify the functionality of continuous writing and reading operations by decrementing the write address from a random starting point while incrementing the read address, all while decrementing the strobe pattern (`1000`, `0100`, `0010`, `0001`). In this test, data is written to random locations and read from random locations, with the strobe pattern decreasing at each step. The test is considered successful if the data written to each address is correctly read back and matches the expected values. Any failures or discrepancies can be identified and analyzed in the scoreboard.

### 4.45.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.45.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.45.4: Additional Arguments

**"run\_op :**

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=decr",

+RD\_ADDR=incr" ,

+LOC=Aldcr"

"owner: bappi"

## 4.46: Bulk write, bulk read: (bulk\_wr\_rd\_byte\_AlDecr\_test)

### 4.46.1: Test Purpose

The purpose of this test is to evaluate the performance and correctness of bulk write and read operations using a strobe pattern that decreases from a random starting point. This test ensures that the system can handle large amounts of data being written and read in a single operation, verifying the efficiency and reliability of bulk data transfers. During the bulk test, all variables are generated randomly, and the memory size can be increased using KNOBS at runtime. The test will pass if all bulk data written is correctly read back and matches the expected values. Any failures or discrepancies can be identified and analyzed in the scoreboard.

### 4.46.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.46.3: Sequence Flow

     1. `cont\_wr\_rd\_knobs\_loc\_seq`

         1.1. `apb\_reset\_seq`

         1.2. `apb\_directed\_seq` (wr\_seq)

         1.3. `apb\_directed\_seq` (rd\_seq)

### 4.46.4: Additional Arguments

**run\_op :**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

"run\_op : +UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=100",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=0",

+WR\_ADDR= ",

+RD\_ADDR= " ,

+LOC=RandL1",

owner  :  bappi"

## 4.47: Continuous write (Address decrement), continuous read (Address decrement): (c\_wr\_rd\_decr\_byte\_Aldecr\_test)

### 4.47.1: Test Purpose

The purpose of this test is to verify the functionality of continuous writing and reading operations by decrementing both the write and read addresses from random starting points, while also decrementing the strobe pattern (`1000`, `0100`, `0010`, `0001`). In this test, data is written to random locations and read from random locations, with the strobe pattern decreasing at each step. The test is considered successful if the data written to each address is correctly read back and matches the expected values. Any failures or discrepancies can be identified and analyzed in the scoreboard.

### 4.47.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.47.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.47.4: Additional Arguments

"run\_op :

+WORD\_KNOB=100",

+HALF\_WORD\_KNOB=0"

+BYTE\_KNOB=0",

+WR\_ADDR=decr ",

+RD\_ADDR=decr ",

+LOC=rand",

"owner: bappi",

**Byte All Location Random Serial Tests:**

## 4.48: cont\_wr\_rd\_byte\_ALRandSr\_test

### 4.48.1: Test Purpose:

The purpose of the test is to verify the functionality of continuous writing and reading operations using strobe (0001, 0010, 0100, 1000) for all location random serial. This test ensures that data can be continuously written and then continuously read back correctly, confirming the basic read and write capabilities of the system. The test will pass if the data written to each address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.48.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.48.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.48.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+LOC=ALincr"

+WR\_ADDR= ",

+RD\_ADDR= "

**owner:**

Amit

## 4.49: cont\_wr\_rd\_incr\_byte\_ALRandSr\_test

### 4.49.1: Test Purpose

The purpose of the test is to ensure proper functionality when continuously writing with address increment and then continuously reading with address increment using strobe (0001, 0010, 0100, 1000) for all location random serial. This test checks that the system can handle sequential addressing, confirming that data is correctly written to and read from successive memory locations. The test will pass if the data written to each incremented address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.49.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.49.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.49.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=incr ",

+RD\_ADDR=incr " ,

+LOC=ALincr"

**owner:**

Amit

## 4.50: cont\_wrIncr\_rdDecr\_byte\_ALRandSr\_test

### 4.50.1: Test Purpose

The purpose of the test is to test the system's ability to handle continuous writing with address increments and continuous reading with address decrements using strobe (0001, 0010, 0100, 1000) for all location random serial. This test ensures that the system can correctly write data to successive memory locations and read back the data in reverse order, verifying the robustness of address handling. The test will pass if the data written to each incremented address is correctly read back in reverse order and matches the expected values. Failures can be checked in the scoreboard.

### 4.50.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.50.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.50.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=incr ",

+RD\_ADDR=decr " ,

+LOC=ALincr"

**owner:**

Amit

## 4.51: cont\_wrDecr\_rdIncr\_byte\_ALRandSr\_test

### 4.51.1: Test Purpose

The purpose of the test is to verify the system's behavior during continuous writing with address decrements and continuous reading with address increments using strobe (0001, 0010, 0100, 1000) for all location random serial. This test checks that the system can correctly write data to memory locations in reverse order and read back the data sequentially, verifying proper functionality of address handling. The test will pass if the data written to each decremented address is correctly read back in sequential order and matches the expected values. Failures can be checked in the scoreboard.

### 4.51.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.51.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.51.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=decr ",

+RD\_ADDR=incr " ,

+LOC=ALincr"

**owner:**

Amit

## 4.52: bulk\_wr\_rd\_byte\_ALRandSr\_test

### 4.52.1: Test Purpose

The purpose of the test is to test the performance and correctness of bulk write and read operations using strobe (0001, 0010, 0100, 1000) for all location random serial. This test ensures that the system can handle large amounts of data being written and read in a single operation, verifying the efficiency and reliability of bulk data transfers. During the bulk test, all variables are generated randomly, and the memory size can be increased using KNOBS at runtime. The test will pass if all bulk data written is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.52.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.52.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.52.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+BYTE\_KNOB=100",

+LOC=ALincr",

+WR\_ADDR= ",

+RD\_ADDR= "

**owner:**

Amit

## 4.53: cont\_wr\_rd\_decr\_byte\_ALRandSr\_test

### 4.53.1: Test Purpose

The purpose of the test is to ensure proper functionality during continuous writing and reading operations with address decrements using strobe (0001, 0010, 0100, 1000) for all location random serial. This test verifies that the system can correctly handle data writes and reads when the address decreases, confirming the system's ability to manage reverse addressing. The test will pass if the data written to each decremented address is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.53.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.53.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.53.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=decr ",

+RD\_ADDR=decr " ,

+LOC=ALincr"

**owner:**

Amit

**#### Byte Random One Location (RandL1) Tests:**

## 4.54: cont\_wr\_rd\_byte\_RandL1\_test

### 4.54.1: Test Purpose

The purpose of this test is to verify the functionality of continuous write and read operations for byte data at a random single location using strobe values `0001`, `0010`, `0100`, or `1000`. This test ensures that the system can handle byte-level transactions at any random location, confirming the flexibility and reliability of single-byte data transfers. The test will pass if the byte data written to a randomly chosen location is correctly read back and matches the expected value. Failures can be checked in the scoreboard.

### 4.54.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.54.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.54.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR= ",

+RD\_ADDR= " ,

+LOC=RandL1"

**owner:**

Amit

## 4.55: cont\_wr\_rd\_incr\_byte\_RandL1\_test

### 4.55.1: Test Purpose

The purpose of this test is to verify the functionality of continuous write and read operations with address increment for byte data at a random single location using strobe values `0001`, `0010`, `0100`, or `1000`. This test checks the system's ability to handle sequential byte-level transactions at any random location, ensuring proper data handling and memory addressing. The test will pass if the byte data written to incremented addresses is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.55.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.55.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.55.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=incr",

+RD\_ADDR=incr" ,

+LOC=RandL1"

**owner:**

Amit

## 4.56: cont\_wrIncr\_rdDecr\_byte\_RandL1\_test

### 4.56.1: Test Purpose

The purpose of this test is to verify the functionality of continuous write operations with address increment and read operations with address decrement for byte data at a random single location using strobe values `0001`, `0010`, `0100`, or `1000`. This test ensures that the system can handle mixed addressing modes for byte-level transactions at any random location, testing the robustness of data management and address sequencing. The test will pass if the data written to incremented addresses is correctly read back from decremented addresses and matches the expected values. Failures can be checked in the scoreboard.

### 4.56.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.56.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.56.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=incr",

+RD\_ADDR=decr" ,

+LOC=RandL1"

**owner:**

Amit

## 4. 57: cont\_wrDecr\_rdIncr\_byte\_RandL1\_test

### 4.57.1: Test Purpose

The purpose of this test is to verify the functionality of continuous write operations with address decrement and read operations with address increment for byte data at a random single location using strobe values `0001`, `0010`, `0100`, or `1000`. This test evaluates the system's capability to handle reverse addressing for write operations and sequential addressing for read operations, ensuring proper byte-level data integrity. The test will pass if the data written to decremented addresses is correctly read back from incremented addresses and matches the expected values. Failures can be checked in the scoreboard.

### 4.57.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.57.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.57.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=decr",

+RD\_ADDR=incr" ,

+LOC=RandL1"

**owner:**

Amit

## 4.58: bulk\_wr\_rd\_byte\_RandL1\_test

### 4.58.1: Test Purpose

The purpose of this test is to test the performance and correctness of bulk write and read operations for byte data at a random single location using strobe values `0001`, `0010`, `0100`, or `1000`. This test ensures that the system can efficiently manage large amounts of byte-level data being transferred to and from any random location, verifying the system's handling of bulk transactions. During the bulk test, all variables are generated randomly, and the memory size can be increased using KNOBS at runtime. The test will pass if all bulk data written to the randomly chosen location is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.58.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.58.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

## 4.58.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+BYTE\_KNOB=100",

+LOC=RandL1",

+WR\_ADDR= ",

+RD\_ADDR= "

**owner:**

Amit

## 4.59: cont\_wr\_rd\_decr\_byte\_RandL1\_test

### 4.59.1: Test Purpose

The purpose of this test is to verify the functionality of continuous write and read operations with address decrement for byte data at a random single location using strobe values `0001`, `0010`, `0100`, or `1000`. This test assesses the system's ability to manage byte-level data transactions while the address decreases, ensuring proper reverse addressing and data integrity. The test will pass if the data written to decremented addresses is correctly read back and matches the expected values. Failures can be checked in the scoreboard.

### 4.59.2: Sequences Used

The main sequence used in this test is `cont\_wr\_rd\_knobs\_loc\_seq`, derived from `apb\_base\_seq`. It includes handles for `apb\_reset\_seq` (as `rst\_seq`) and `apb\_directed\_seq` (as `wr\_seq` for write operations and `rd\_seq` for read operations). This sequence first performs a reset using `rst\_seq`, followed by write operations with `wr\_seq` and read operations with `rd\_seq`. The write and read transactions are designed to test the APB slave with different strobe values and address patterns. The strobe signals and addresses are generated based on plusargs (`WORD\_KNOB`, `HALF\_WORD\_KNOB`, `BYTE\_KNOB`, `WR\_ADDR`, `RD\_ADDR`, `LOC`) provided at runtime. The strobe values are controlled by the `weighted\_strb\_c` constraint, ensuring varied data widths, while the address generation method (`addr\_gen()`) ensures sequential or random address patterns as specified. This setup provides a flexible and comprehensive verification of the APB slave.

### 4.59.3: Sequence Flow

1. `cont\_wr\_rd\_knobs\_loc\_seq`

1.1. `apb\_reset\_seq`

1.2. `apb\_directed\_seq` (wr\_seq)

1.3. `apb\_directed\_seq` (rd\_seq)

### 4.59.4: Additional Arguments

**run\_op:**

+UVM\_TESTNAME=c\_wr\_rd\_knobs\_loc\_test",

+UVM\_TIMEOUT=500000",

+UVM\_VERBOSITY=UVM\_HIGH",

+WORD\_KNOB=0",

+HALF\_WORD\_KNOB=0",

+BYTE\_KNOB=100",

+WR\_ADDR=decr",

+RD\_ADDR=decr",

+LOC=RandL1"

**owner:**

Amit

4.60: ALL Random Test

# 5.Environment

An environment provides a well-mannered hierarchy and container for agents, scoreboards, and other verification components including other environment classes that are helpful in reusing block-level environment components at the SoC level. The user-defined env class has to be extended from the uvm\_env class.

**Fields:**

1.environment\_config env\_cfg

2.apb\_agent apb\_agnt

3.apb\_agent\_config apb\_agnt\_cfg

|  |  |
| --- | --- |
| **Methods** | **Description** |
| **1.new** | Constructor method. As apb\_env is a component it’s constructor has two argumnets string name and uvm\_component parent. |
| **2.build\_phase** | Virtual method, return type void. environment\_config values have been accessed by uvm\_config\_db and then agent configuration values have been assigned with the values that we set from test by aggregating environment config. Then apb\_agent\_config has also been set. Finally, apb\_agent has been created. According to the status of environment config’s has\_scb, a scoreboard will be created. |
| **3.connect\_phase** | Virtual method, return type void. In the connect phase the connection between scoreboard, coverage and monitor will be made |

In this testbench first apb\_env has been extended from uvm\_env and then registered with the factory accordingly. Then the handles of environment\_config, apb\_agent and apb\_agent\_config have been taken. In the build phase environment\_config values have been accessed by uvm\_config\_db and then agent configuration values have been assigned with the values that we set from test by aggregating environment config. Then apb\_agent\_config has also been set. Finally, apb\_agent has been created. According to the status of environment config’s has\_scb , a scoreboard will be created.

## 5.1: Environment Config

**Fields:**

1.uvm\_active\_passive\_enum apb\_agent\_stat

2.bit apb\_agent\_cover

3.bit has\_scb

4.bit apb\_mntr\_log

**Methods:**

**1.new:** Constructor method. As it of object type, it only has one argument i.e string name.

The environment\_config class has been extended from uvm\_object and then registered with factory accordingly. Here a built in uvm\_active\_passive\_enum variable named apb\_agent\_stat, has been declared and assigned with default values 0 which means if this is not assigned 1, the agent will be considered as passive agent. In the same way a bit type variable have been decaled named apb\_agent\_cover assigned with 0 which means by default there will be no coverage class. Bit has\_scb has also been decaled and assigned 0 which denotes no scoreboard in the environment class by default. Bit apb\_mntr\_log has been declared so that the generation of log file from monitor can be controlled.

## 5.2: Agent

Agent is extended from uvm\_agent. An agent contains driver, sequence and monitor. The verification components - monitor, collector, and sequencer are all instantiated by the agent. It also uses TLM connections to bind these components. The agent can operate in either active or passive mode. The agent instantiates the driver, sequencer, collector, and monitor in the active mode of operation, while only the monitor and collector are instantiated and configured in the passive mode of operation.

**Fields:**

1.apb\_agent\_config apb\_agnt\_cfg

2.apb\_sequencer apb\_sqcr

3.apb\_driver apb\_drvr

4.apb\_monitor apb\_mntr

5.uvm\_analysis\_port apb\_agnt\_port

**Methods:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| **1.new** | Constructor function. As agent is a component it has two arguments in its’ constructor function. |
| **2.build\_phase** | Virtual method, return type void. In the build phase first apb\_agent\_config’s information are accessed using uvm\_config\_db and then depending upon the information such as agent is active or passive, sequencer,driver and coverage is created. The monitor and apb\_agnt\_port have also been created here. |
| **3.connect\_phase** | Virtual method, return type void. In the connect phase the connection between driver and sequencer TLM port has been made depending upon the value of apb\_agnt\_cfg. If agent is active the value of apb\_agnt\_cfg will be 1 and thus driver and sequencer TLM connection will be made. The monitor and apb\_agnt analysis port is also connected. |

In this testbench apb\_agent.sv file is used for agent. Here apb\_agent is extended from uvm\_agent and registered with factory. Handles of apb\_agent\_config, apb\_sequencer,apb\_driver and apb\_monitor have been taken. An analysis port named apb\_agnt\_port is taken which is connected to the monitor analysis port later so that we can control the monitor port from higher level component agent. In the build phase first apb\_agent\_config’s information are accessed using uvm\_config\_db and then depending upon the information such as agent is active or passive, sequencer,driver and coverage is created. The monitor and apb\_agnt\_port have also been created here. In the connect phase the connection between driver and sequencer TLM port has been made. The monitor and apb analysis port is also connected.

### 5.2.1: APB Agent Config

Here apb\_agent\_config is extended from uvm object. A variable of uvm\_active\_passive\_enum is taken named is\_active and is denoted a value UVM\_PASSIVE. A bit type variable has\_cover is declared and assigned with 0 which denotes there will be no coverage.

**Fields:**

1.uvm\_active\_passive\_enum is\_active

2.bit has\_cover

3.bit apb\_mntr\_log

**Methods:**

**1.new:** Constructor function. As it is of object type the constructor function has only one argument i.e string name.

### 5.2.2: APB Defines

**apb\_defines.sv** SystemVerilog file is used to parameterize the ADDR\_WIDTH as 5, DATA\_WIDTH as 32 and BYLE\_LANE which is used for strobe write as 4 with `define. From now on where there is any need to define the size, we will simply pass these parameters.

### 5.2.3: APB Interface

In apb\_interface.sv file first an interface is taken named apb\_interface where all the top module signal is declared with type logic.

**Fields:**

bit PCLK\_i

bit PRESETn\_i

logic[`ADDR\_WIDTH-1:0] PADDR\_i

logic PWRITE\_i

logic[`DATA\_WIDTH-1:0] PWDATA\_\_i

logic[`BYTE\_LANE-1:0] PSTRB\_i

logic PSEL\_i

logic PENABLE\_i

logic[`DATA\_WIDTH-1:0] PRDATA\_o

logic PREADY

logic PSLVERR\_o

**Clocking Blocks:**

Two clocking blocks are declared. The clocking block for driver is cb\_drvr and the block for monitor is cb\_mntr. In the driver clocking block, all the input ports of the dut are declared as output and all the output ports are declared as input. In the monitor clocking block, all the ports are declared as input. Two modport blocks mp\_drvr and mp\_mntr have been added where the PRESETn\_i signal is added with corresponding clocking block.

### 5.2.4: Sequence item

**apb\_seq\_item.sv** file is created for serving the purpose of sequence item. The apb\_seq\_item is expanded from uvm\_sequence\_item.

**Fields:**

rand bit PCLK\_i

rand bit PRESETn\_i

rand bit[`ADDR\_WIDTH-1:0] PADDR\_i

rand bit PWRITE\_i

rand bit[`DATA\_WIDTH-1:0] PWDATA\_\_i

rand bit[`BYTE\_LANE-1:0] PSTRB\_i

rand bit PSEL\_i

rand bit PENABLE\_i

bit [`DATA\_WIDTH-1:0] PRDATA\_o

bit PREADY

bit PSLVERR\_o

int item\_count

The inputs such as PRESETn\_i,PSEL\_i,PADDR\_i,PWRITE\_i etc will be generated randomly so added rand modifier before all the input fileds.

**Factory Registration:**

All the data members are registered with factory with the flags UVM\_ALL\_ON. Along with this flag PADDR\_i, PWDATA\_i and PRDATA\_o are registered with UVM\_DEC,UVM\_HEX,UVM\_HEX flag respectively so that these data are shown in specified format. As we are adding these field macros to data members, we will be able to use the core methods of the uvm such as print, copy, clone etc.

**Fields for knob variables:**

int WORD\_KNOB

int HALF\_WORD\_KNOB

int BYTE\_KNOB

To pass the knob value from command line three int type variables named WORD\_KNOB, HALF\_WORD\_KNOB, BYTE\_KNOB have been taken.

**Methods:**

**1.new:** Constructor function. In the constructor function using $value$plusargs they are assigned to the value that are set from the command line otherwise they will be assigned to the default values.

**Constraint:**

Two constraints have been taken. They are strb\_knob and address. In the strb\_knob the probability of generating byte, half word and word type sequence item has been defined. In the address constraint, the address values are taken as the multiple of 4 which makes these address values aligned.

Lastly apb\_valid\_addr\_item is extended from apb\_seq\_item. It is registered with the factory accordingly .In the constructor function new the argument string name has been passed. It has a constraint named valid\_addr where PADDR\_i is less than the value of 1024.

### 5.2.5: Sequencer

The UVM sequencer serves as a link between the driver and the sequence. It sends the sequences to the driver. It also serves as an arbitrator for running many sequences in parallel. Sequencer regulates the flow between sequence and driver. The user uses TLM port to connect between driver and sequencer. The default TLM for sequencer is seq\_item\_export which is connected to driver’s default TLM port seq\_item\_port in the agent class. Usually there is no need to extend the sequencer from uvm\_sequencer instead the built in implementation of uvm\_sequencer is used. In this testbench **apb\_sequencer.sv** file has been used where **apb\_sequencer** is extended from uvm\_sequencer and then registered it in factory and added constructor accordingly. As it is a component there are two arguments in the constructor function.

### 5.2.6: Sequence

A collection of transactions is referred to as a sequence. A sequence is created by extending the uvm\_sequence. It generates a series of sequence items and sends them to the driver via sequencer. Users in the sequence class will generate complex stimuli. These sequences can be mixed, randomized, and expanded to generate new sequences. **apb\_seq\_lib.sv** file is created for generating different sequence as per the test plan. The message of start and end of the sequence is parameterized as SEQ\_SP and SEQ\_EP at the beginning of this file.

#### 5.2.6.1: Base Sequence

At first **apb\_base\_seq** is expanded from the uvm\_sequence and the parameter passed here is apb\_seq\_item as it will be doing transaction of type apb\_seq\_item. As it is an object, we register this to a factory and added constructor function new accordingly.

**Fields:**

bit [`ADDR\_WIDTH-1:0] addr

bit op

bit[`DATA\_WITHD-1:0] data

bit[`BYTE\_LANE-1:0] strb

**Methods:**

**body:** It is avirtual task. In the body task a reporting macro has been added.

#### 5.2.6.2: Write Sequence

From the apb\_base\_seq, **apb\_write\_seq is extended.** In the body task default handle of apb\_seq\_item named req is randomized keeping PWRITE\_i and PRESETn\_i signal high.

#### 5.2.6.3: Read Sequence

apb\_read\_seq has been extended from the apb\_base\_seq. In the body task default handle of apb\_seq\_item named req is randomized keeping PWRITE\_i and PSTRB\_n as low and PRESETn\_i signal high.

#### 5.2.6.4: Reset Sequence

apb\_reset\_seq has been extened from apb\_base\_seq. In the body task default handle of apb\_seq\_item named req is randomized keeping PWRITE\_i high and PRESETn\_i signal low.

#### 5.2.6.5: Random Sequence

apb\_rand\_seq has been extended from apb\_base\_seq. In the body task default handle of apb\_seq\_item named req is randomized keeping PRESETn\_i high which means all other values will be generated randomly.

#### 5.2.6.6: Directed Sequence

apb\_directed\_seq is extended from apb\_base\_seq. In the body task default handle of apb\_seq\_item named req’s value is manually assigned with the local variables of apb\_base\_seq.

### 5.2.7:Driver

**Fields:**

1.virtual apb\_interface apb\_dintf

2.apb\_seq\_item item\_obj

**Methods:**

|  |  |
| --- | --- |
| **Methods** | **Description** |
| **1.new:** | Constructor Function. As driver is a component it has two arguments in the constructor function. |
| **2. delay\_assign** | Virtual task. It has an argument of apb\_interface. In this task a single clock cycle delay has been taken. |
| **3. sel\_enb\_change** | Virtual task. There is an argument of virtual apb\_interface**.** In this task first PSEL\_i has been driven 1. Then delay\_assign task is called which will add one clock cycle delay and then PENABLE\_i is driven 1. |
| **4. contr\_signl\_chng** | Virtual task. An argument of apb\_seq\_item and virtual apb\_interface is taken**.** In this task data signals such as PADDR\_i, PDATA\_i and control signals PWRITE\_i, PSTRB\_i are driven with the value of item. |
| **5. drive** | Virtual task. There is an argument of apb\_seq\_item. First an int type variable named cycle\_count is declared which will count the delay of getting PREADY signal as high from apb slave. Then after a delay of one negative clock edge in the fork join two tasks sel\_enb\_chage, contr\_signl\_chng are called. As per the protocol PENABLE\_i needs to be high right after one clock cycle of making PSEL\_i high. A do while loop is uded to see whether we are getting PREADY\_o from the slave or not. It will search for the PREADY\_o signal at the positive edge of the PCLK\_i for 20 clock cycle. After each clock cycle the value of cycle\_count will be incremented by one. If PREADY\_o does not come in this specified time it will throw a fatal. If PREADY\_i is 1 then PSEL\_i and PENABLE\_i will be assigned zero. |
| **6. reset\_intf** | Virtual task. In the reset\_intf task first the **PRESETn\_i** signal is kept low along with the other input signals which will reset the system as PRESETn\_i is a active low signal. After one clock cycle the PRESETn\_i is deasserted i.e it is made it high so that the system can be in operational mode. |
| **7.run\_phase** | Virtual task. In the run phase of the driver first driver will request for a sequence from the sequencer by seq\_item\_port.get\_next\_item(req). Upon getting this request grant, sequence will send specific sequences to driver. If the PRESET\_i value of the sequence\_item is 0 then it will drive the reset\_intf task otherwise it will send the drive task to the dut. Finally driver will call seq\_item\_port.item\_done() and can procced for the next sequence |

For error driver sel\_enb\_error\_drive is extended from apb\_driver. The delay\_assign task is overridden and a random clock delay between 0 to 10 will be generated here.

The driver is extended from uvm\_driver. It drives the DUT. The driver starts the next transaction request and sends it down to the lower-level components with the help of DUT. In this testbench **apb\_driver.sv** file has been created where **apb\_driver is** extended from uvm\_driver. As it is a component it is registered with factory and constructor function is added accordingly. An instance of virtual apb\_interface as **apb\_dintf** is taken along with an instance of apb\_seq\_item as **item\_obj**. In the build phase of the driver the access of virtual interface has been got by using uvm\_config\_db with appropriate key value which is “APB\_INTF”. To drive the sequences to dut according to the protocol two tasks named **drive** and **reset\_intf are** defined. In the reset\_intf task first the **PRESETn\_i** signal is kept low along with the other input signals which will reset the system as PRESETn\_i is a active low signal. After one clock cycle the PRESETn\_i is deasserted i.e it is made it high so that the system can be in operational mode. In the drive task first an int type variable named cycle\_count is declared which will count the delay of getting PREADY signal as high from apb slave. Then at the negative edge of PCLK\_i the PADDR\_i,PWRITE\_i,PSEL\_i,PSTRB\_i signals are driven. As per the protocol PENABLE\_i needs to be high right after one clock cycle of making PSEL\_i high. So made PENABLE\_i high after waiting for one clock cycle at the negedge of PCLK\_i . Then a do while loop is uded to see whether we are getting PREADY\_o from the slave or not. It will search for the PREADY\_o signal at the positive edge of the PCLK\_i for 20 clock cycle.After each clock cycle the value of cycle\_count will be incremented by one. If PREADY\_o does not come in this specified time it will throw an error.In the run phase of the driver first driver will request for a sequence from the sequencer by seq\_item\_port.get\_next\_item(req). Upon getting this request grant, sequence will send specific sequences to driver. If the PRESET\_i value of the sequence\_item is 0 then it will drive the reset\_intf task otherwise it will send the drive task to the dut. Finally driver will call seq\_item\_port.item\_done() and can procced for the next sequence.

### 5.2.8: Monitor

A UVM monitor is a passive component used to capture DUT signals using a virtual interface and translate them into a sequence item format. It is derived from uvm\_monitor. These sequence items or transactions are broadcast to other components like the UVM scoreboard, coverage collector, etc. It uses a TLM analysis port to broadcast transactions.

**Fields:**

1.virtual apb\_interface apb\_mintf

2.apb\_seq\_item apb\_mitem

3.uvm\_analysis\_port #(apb\_seq\_item) apb\_mntr\_port

4. int ok

**Fields for Monitor log File:**

1.bit apb\_mntr\_log

2.int file\_handle

3.string log\_filename

4.int start\_time

5.int end\_time

6.int transaction

**Methods:**

|  |  |
| --- | --- |
| **Fields** | **Description** |
| **1.new** | Constructor function. As monitor is a component it has two arguments named string name and uvm\_component parent. |
| **2.set\_log\_file** | Return type void. First log\_filename value is assigned to file\_handle. Then some file header information has been written utilizing $fwrite. |
| **3.log\_file\_write** | **Return type void. Values will be written according to the header in specific column.** |
| **4.build\_phase** | **Virtual function, return type void.** In the build phase virtual interface is accessed using uvm\_config\_db and analysis port is created. Depending upon the value of apb\_mntr\_log ,set\_log\_file is called. Thus a log file for monitor will be created. |
| **5.run\_phase** | **Virtual task.** In the run phase all the signals are sampled at the positive edge of the clock. Then start\_time is assigned with $realtime at the starting of the transaction. If the reset is high, then all other signals will be sampled. Depending upon the value of the PSEL\_i ,item will be created and the value of PADDR\_i ,PWRITE\_i will be sampled in the item. After that if the PWRITE\_i is high then the value of PWDATA\_i and PSTRB\_i will be sampled. Then after waiting for one clock cycle if PENABLE\_i is high ,depending upon the value of PREADY\_i PRDATA\_o will be sampled and then all the sampled information will be broadcasted by the write function of analysis port .If monitor did not sample PENABLE\_i high after one clock cycle of PSEL\_i high, it will throw an error message. |

In this testbench apb\_monitor is extended from uvm\_monitor. Handles of apb\_interface are taken as virtual named apb\_mintf and a handle of apb\_seq\_item is taken as apb\_mitem. Then constructor function is used. An analysis port is also taken for broadcasting the item information which is apb\_mntr\_port. In the build phase virtual interface is got using uvm\_config\_db and analysis port is created. In the run phase all the signals are sampled at the positive edge of the clock. If the reset is high, then all other signals will be sampled. Depending upon the value of the PSEL\_i, item will be created and the value of PADDR\_i ,PWRITE\_i will be sampled in the item. After that if the PWRITE\_i is high then the value of PWDATA\_i and PSTRB\_i will be sampled. Then after waiting for one clock cycle if PENABLE\_i is high ,depending upon the value of PREADY\_i PRDATA\_o will be sampled and then all the sampled information will be broadcasted by the write function of analysis port .If monitor did not sample PENABLE\_i high after one clock cycle of PSEL\_i high, it will throw an error message.

### 5.2.8.1: resetable\_monitor

The **resetable\_monitor** class extends the functionality of the **apb\_monitor** by introducing **reset-awareness** to the monitoring process. Its primary purpose is to handle scenarios where the reset signal (PRESETn\_i) toggles during simulation, which can affect the behavior of the APB interface and transactions. When the reset signal (PRESETn\_i) is asserted (0), the resetable\_monitor disables transaction monitoring. This avoids capturing invalid transactions or misleading data during reset. When the reset signal is deasserted (1), the monitor resumes its normal operation, ensuring only valid transactions are observed and recorded. The class triggers **UVM events** (reset\_on and reset\_off) to notify other verification components (like the scoreboard or coverage collector) about the reset state changes. This synchronization is crucial for components dependent on the reset state. Transactions occurring during or immediately after a reset could be incomplete or erroneous. By monitoring the reset state, the resetable\_monitor ensures such invalid transactions are ignored.

**Fields:**

1.uvm\_event reset\_on: Triggered when PRESETn\_i is asserted.

2 .uvm\_event reset\_off: Triggered when PRESETn\_i is asserted (0).

The run\_phase method uses a fork-join construct to implement two parallel threads. One for monitoring reset signals and triggering events. Another for normal transaction monitoring, which is paused during reset. The resetable\_monitor retains all the core functionality of apb\_monitor while adding reset-awareness, making it a drop-in replacement. The reset\_check thread continuously monitors the reset signal (PRESETn\_i). When reset is asserted (0), it triggers the reset\_on event and disables the monitor\_run thread. When reset is deasserted (1), it triggers the reset\_off event and re-enables the monitor\_run thread.

### 5.2.8: Coverage

The apb\_coverage class is an extension of the uvm\_subscriber class and is designed to collect functional coverage data for an APB (Advanced Peripheral Bus) interface. It uses SystemVerilog's covergroup construct to capture the coverage of various signals and transactions. This class is instantiated and connected to the UVM environment for collecting and reporting coverage metrics.

**Fields:**

1. apb\_seq\_item item

A handle to store the received transaction from the monitor.

2. real cov\_val

Used to store the total coverage percentage calculated from the covergroup.

3. covergroup apb\_cg;

The main coverage group capturing functional coverage data for APB signals. The covergroup includes coverpoints and cross-coverage to ensure all significant scenarios are observed during simulation. Each coverpoint targets a specific APB signal or behavior. They are given below:

STROBE: coverpoint item.PSTRB\_i

Captures the reset behavior with various bins representing low, mid, and high reset levels.

Address (PADDR): PADDR: coverpoint item.PADDR

Splits the address range into two bins: low and high, based on the address width.

Write Data (PWDATA): PWDATA: coverpoint item.PWDATA\_i

Tracks write data values with specific bins for low (all 0s) and high (all 1s).

Read Data (PRDATA): PRDATA: coverpoint item.PRDATA\_o

Captures read data values, splitting them into low and high bins.

Control Signals:

PWRITE: coverpoint item.PWRITE\_i

PSEL: coverpoint item.PSEL\_i

**PWRITE**: Differentiates between write (1'b1) and read (1'b0) operations.

**PSEL**: Indicates if the APB slave is selected (1'b1) or not (1'b0).

**Cross-Coverage**: Combines multiple coverpoints to ensure interactions between signals are covered.

Write Transactions (CROSS\_WRITE): CROSS\_WRITE: cross PADDR, PWDATA, PSEL, PWRITE

* Captures all possible combinations of address, write data, select signal, and write-enable.

**Read Transactions (CROSS\_READ):** CROSS\_READ: cross PADDR, PRDATA, PSEL, PWRITE

* Tracks combinations for address, read data, select signal, and read-enable.

**Ignored Bins**: Exclude irrelevant combinations using ignore\_bins

UVM Phases

Constructor (new)

function new(string name, uvm\_component parent);

Instantiates the coverage group and sets the component name and parent.

Build Phase

function void build\_phase(uvm\_phase phase);

Logs a message indicating that the build phase has started.

Write Method

function void write(apb\_seq\_item t);

Receives a transaction from the monitor and samples it in the covergroup.

Extract Phase

virtual function void extract\_phase(uvm\_phase phase);

Computes the overall coverage percentage using get\_coverage().

Report Phase

virtual function void report\_phase(uvm\_phase phase);

Reports the final coverage value at the end of the simulation

The apb\_coverage class collects detailed functional coverage metrics for the APB protocol signals and transactions, enabling thorough validation of the APB interface. This coverage data ensures that the design has been exercised across all expected scenarios and corner cases.

## 5.3 Scoreboard:

Scoreboard is extended from uvm\_scoreboard. The UVM scoreboard is a component that checks the functionality of the DUT. It receives transactions from the monitor using the analysis export for checking purposes. It compares expected value and actual value. The scoreboard has a reference model to compare with design behavior. The reference model is also known as a predictor that implements design behavior so that the scoreboard can compare DUT outcome with reference model outcome for the same driven stimulus.

**Fields:**

1.apb\_seq\_item read\_trans[$]

2.apb\_seq\_item rcvd\_item, mtch\_item

3.uvm\_analysis\_imp mtr2scb\_impport

4.logic [31:0] data\_memory[logic [31:0] ]

5. uvm\_event reset\_assert, reset\_deassert

**Methods:**

|  |  |
| --- | --- |
| **Fields** | **Description** |
| **1.function new ()** | Constructor function. As scoreboard is a component it has two arguments named string name and uvm\_component parent. |
| **2.virtual function void write (apb\_seq\_item received\_item)** | **Virtual function. Return type void.** The write function enables the scoreboard to capture, store, and analyze transactions, allowing it to fulfill its role of verifying the design's correctness. First the instance of rcvd\_item is created. To avoid any mismatch from monitor at a time being, the received\_item is casted to rcvd\_item. If rcvd\_item.PRESETn\_i is 0 then the existing data\_memory will be deleted else depending upon the value of PSLVERR, pt\_err and PWRITE\_i ,slave\_error, protocol\_error, memory\_write or memory\_read function will be called. |
| **3.Virtual function void slave\_error(apb\_seq\_item err\_item)** | Virtual function.Retrun type void. If during read transaction i.e err\_item.PWRITE\_i=0 there is any strobe signal i.e err\_item.PSTRB\_i!=0 then an error message will be printed. |
| **4.virtual function void memory\_write(apb\_seq\_item wr\_item)** | Virtual function. Return type void. The first three variables of bit type of length 32 bit named existing\_data, new\_data and final\_data have been taken. Then the current data of data\_memory at address wr\_item.PRADDR\_i is assigned into existing\_data and data PWDATA\_i is assigned into new\_data. Then depending upon the strobe signal PSTRB\_i existing\_data and new\_data will be updated. And finally updated data will be wriiten into data\_memory. |
| **5.virtual function void memory\_read(apb\_seq\_item rd\_item)** | Records read transactions to the read\_trans queue for future comparison  Checks if the address exists in data\_memory. If not, it logs a warning about an attempt to read an empty address. |
| **6.virtual function build\_phase(uvm\_phase phase)** | Part of the UVM build phase, initializes mntr2scb\_impport (a communication port for the scoreboard). his enables the scoreboard to receive data items sent by other components (e.g., monitors). |
| **7.virtual task run\_phase(uvm\_phase phase)** | Continuously checks for read transactions that need verification by calling data\_match in an infinite loop. This ensures that every read transaction in read\_trans is eventually verified |
| **8.virtual task data\_match** | Compares the expected data in data\_memory with the read data from the transaction and logs if they match or not. Waits until read\_trans has entries, then retrieves the oldest entry (mtch\_item). It compares the PRDATA\_o in mtch\_item with the data stored in data\_memory for the same address (PADDR\_i). If they match, it logs an info message; if not, it logs an error with the mismatched data details. |
| **9. virtual function protocol\_error(apb\_seq\_item err\_item);** | Handles protocol violations for write or read operations, logging a message and processing the erroneous transaction. |
| **10. virtual function void check\_phase(uvm\_phase phase);** | Ensures the read\_trans queue is empty at the end of simulation, logging an error if it’s not. |